JOB-SHOP SCHEDULING

by

BALRAJ SINGH SONDHU

B. Sc. (Mechanical Engineering),
Panjab University, 1964

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Industrial Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1967

Approved by:

L.E. Grosh
Major Professor

TABLE OF CONTENTS

rare. In practice the majority of job-shop production orders are executed only once, and only a small percentage of them are repeated regularly or intermittently.

A job-shop production schedule that establishes the starting and finishing dates (times) of all jobs is subject to the limitations of the availability of the following:

1. Facilities of the type required to process the jobs being scheduled.
2. Operators who possess the desired skill and experience to operate the equipment and perform the type of work involved.
3. Necessary materials and purchased parts, if any.

The inadequacy of any one of these factors, or at least faulty knowledge as to when the missing factors will be made available, prevents the development of an intelligent production schedule. In a job-shop we try to match the requirements set out in a production order (part number, quantities, dates of delivery), with the available men, machines, and materials. There may be several ways in which the requirements can be met, i.e., several ways of routing the jobs through machines. We have to find the schedule which is best according to some predetermined criteria (measure of effectiveness). Usually this criteria is based on costs or profits. The solution that insures the attainment of the objectives of a production order at lowest possible costs, such that long term profits are maximized, is considered to be the best one.

The first job is to find if a feasible schedule exists. If there is more than one feasible schedule, then we may choose an optimal one. In the optimization process the criteria used is known as an objective function. Thus the job-shop scheduling problem is essentially a problem of determining the order (sequence) of jobs on different machines so that some objective function (e.g., total elapsed time, average idle time on machines, manhours, expected profit, expected cost, etc.) is optimized. In other words, the problem of scheduling is one of sequencing jobs for each machine.

Job-shop scheduling is a dynamic process. Schedules cannot be set once for all times. The schedule of every new order must be integrated with the processing of jobs already in partial stages of completion.

Mathematical analysis of the scheduling problem has recently begun. Some progress has been made to date. The formulation of the problem itself is incomplete because it is concerned with minimizing some function of time. The balencing of conflicting objectives has not yet been brought into the scheduling problem. For example, in scheduling jobs over a series of machines, we are not only concerned with minimizing total elapsed time (in order to reduce the cost of in process inventory and to increase output for fixed investment) but usually are also concerned with providing equal incentive opportunities to operators of different machines. These and other requirements such as shipping priorities and delivery dates are generally in conflict with the objective of minimizing some

function of processing time.

The first step toward scheduling was the use of the Gantt charts. Useful as these charts are, they often fail to yield optimum schedules or to indicate how far their output is from an optimum schedule. Researchers have devised algorithms to improve the use of the Gantt charts. By the term algorithm we mean a formal set of logical rules for the computation of some desired criteria. Four different approaches to the problem will be discussed, namely, (1) graphical, (2) combinatorial, (3)integer linear programming, (4) schedule algebras.

For the better application of the scheduling procedures, one should be able to obtain: (1) Good approximations of times for the jobs on different machines; (2) the desired sequence of machines to process the job, and any possible alternate sequences.

1.1. The Gantt Chart

The principles of Gantt charting were laid down by Henry L. Gantt, a pioneer in scientific management movement. Though this chart is named after Gantt, he was not the first to use the charts to illustrate production situations but he was the first to put his idea in writing. This was the first attempt at visual control of the machine loading and production process. It attempted to correlate machines or manufacturing orders or material versus time. In other words, the basic principle used is that the work planned and the work accomplished are shown on the same chart in relation to each other and also in their relation to time. The items are listed in a column, with corresponding capacities or data on maximum scheduling loads shown in the adjacent column. Other columns are used for time units, such as hours, days, weeks, or months. In the chart a unit space portrays both an amount of time and work to be done in that time.

Figure 1.1 shows a Gantt chart representing the scheduling of work for a department in which four machines, A, B, C, and D, are used. A main time column has five divisions representing five working days in the week. To illustrate, the column headed "July 3" means the week ending July 3. The straight lines drawn horizontally represent the amount of work. For each machine, the work scheduled by weeks is indicated by the light line and total cumulative work scheduled by the heavy line. Thus for machine D, work scheduled for the week ending July 10 is four days, which represents 60 items; and total amount of time scheduled for this machine for five weeks' work

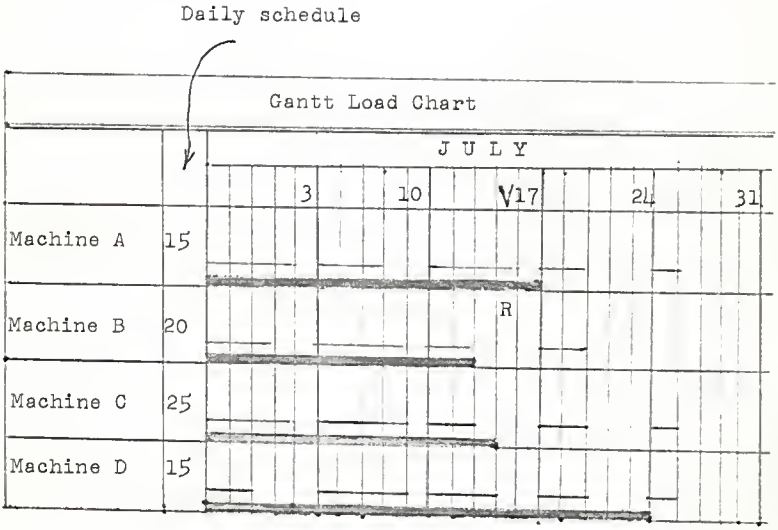

Fig. 1.1. Gantt load chart showing graphically the degree of utilization of machines, and idle time available for scheduling.

is twelve days. The V mark on the top of the chert shows that the chart represents the status of that date, which in the illustration is July 16. This type of Gantt chart is termed e load chart because it represents the load assigned to each machine. If there are any letters under the horizontal lines, they indicate the reason for any delay. For instance 'R' in the row of machine A indicates the delay due to repairs.

The edvantages of a Gentt chart as listed by Wallace Clark are:

1. The use of a Gantt chart makes it necessary to plan. The plan is presented so clearly that it can be understood in detail.
2. It compares what is done with what was done, thus indicating the progress made, and, if progress is not satisfectory, it tells the reason why.
3. It fixes the responsibility for the success or failure of a plan. Causes and effects with their relation to time are brought out so clearly that it becomes possible for executives to foresee future happenings.
4. It is remarkably compact. There is continuity which emphasizes any break in records or any lack of knowledge as to what has taken plece.
5. It is easy to construct. No drafting experience is necessary.
6. It is easy to read.
7. It visualizes the passing of time and thereby helps to reduce idleness and waste of time.

8. It presents facts in relation to time and is therefore dynamic. The chart becomes a moving force for action.

The Gantt chart has the following disadvantages:

1. Once a production schedule has been fixed it becomes difficult to introduce any changes in it.
2. The chart does not easily tell the standing of a particular production order at any moment in time.
3. It does not optimize any objective functions, e.g., the total time required for processing of the jobs in a job-shop. The main reason for this being the limit to an individual's memory. This has led the researchers to devise algorithms which minimize some objective functions under certain assumptions.

The algorithms for solving the job-shop scheduling problems make it possible to use computers and quickly redo the scheduling whenever any change in the situation occurs. Most of the procedures amplify the use of the Gantt chart. In the next section we will see how the graphical concept of the scheduling problem can be used to minimize the make-span, i.e., the total time required to complete the processing of a given number of jobs on a given set of facilities.

2.0. GRAPHICAL APPROACHES TO THE PRODUCTION SCHEDULING PROBLEM

S. Akers and J. Friedman (2) considered in 1955 the following problem:

"Given n parts to be fabricated on m machines. The order of scheduling each part through the machines and operating time of each part on each machine are known. The problem is to find the sequence of fabricating parts so that the total elapsed time to complete the manufacture of all parts is minimal." The method of solution for the case of two parts and m machines is given.

2.1. The Case of Two Jobs on m Machines

In 1956 Akers (3) gave a graphical approach for solving the above problem involving two jobs and m machines. He took the same problem discussed in (2, 14). Let us take another example: there are four machines (A, B, C, and D) and two jobs, job 1 and job 2. Job 1 is A_3 B_2 C_4 D_5 and job 2 is C_3 B_2 A_4 D_5. Subscripts denote time in hours required for each operation. Take job 1 on x-axis and job 2 on y-axis. Shade the rectangles defined by corresponding operations on the two axes. We draw a continuous line joining the origin with point P, this line having straight-line segments which are horizontal, vertical, and diagonal (45 degrees). The origin and point P are the start and finish points, respectively. The other points in the outer rectangle show degrees of completion. Any points in the shaded area are infeasible. A point in the shaded area means

that both of the jobs are being processed simultaneously on the same machine. Thus shaded areas are regions of infeasibility. Paths in the rectangle form (0, 0) to P, which do not pass through infeasible regions, correspond to feasible sequence. In a path, horizontal movement corresponds to the processing of job 1, vertical movement to the processing of job 2, and diagonal movement to the processing of both jobs simultaneously. An optimal feasible path and an infeasible path are shown in Fig. 2.1. If a path passes below the infeasible region corresponding to a machine, the sequence corresponding to this path will have job 1 processed first on this machine; if it is above, job 2 is processed first on this machine and we put a bar over the corresponding letter; $a\bar{b}$ thus means on Machine A job 1 is processed first and on Machine B job 2 is processed first. The shortest of such paths would represent an optimal program.

In finding the shortest line, however, we must consider the projections (on an axis) of the diagonal segments. Thus length of a program is Σ(vert. segments) + Σ(horiz. segments) + $\frac{1}{\sqrt{2}}\Sigma$(45-degree segments).

From Akers-Friedman feasibility theorem I[1] it follows that a, $\bar{b}$, $\bar{c}$, d is feasible. In the above example the shortest line is 20 hours. This method offers very quick solutions for the 2 x m case. Nothing is done about the general case.

Szwarc (81) has solved the same problem (2 x m) by a

---

[1]Akers and Friedman feasibility theorem I states: A necessary and sufficient condition that a 2-job program be technologically feasible is that for each pair of machines X and Y where X precedes Y for job 1 and X follows Y for job 2, the term $\bar{x}y$ does not appear in the program.

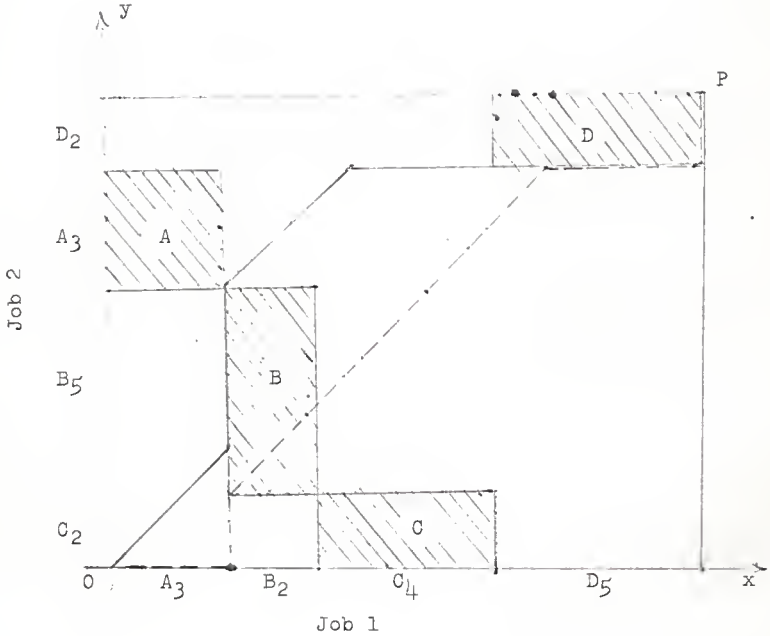

Fig. 2.1. The dotted line corresponds to an infeasible schedule.

method which is a combination of dynamic programming and graphical approach. The problem is to find the sequence of processing the parts so that the total elapsed time to complete the manufacture of all the parts is minimized. Even for large 'm' the solution can be quickly obtained.

Let t_{ij} be the processing time for job i (i = 1, 2) on machine j(j = 1, 2, . . ., m). Let $(r_1, r_2, \ldots, r_m)$ represent the processing sequence for job 1 and $(e_1, e_2, \ldots, e_m)$ that for job 2; where $(r_1, r_2, \ldots, r_m)$ and $(e_1, e_2, \ldots, e_m)$ are permutations of numbers (1, 2, . . ., m).

One assumption is made here; i.e.,

$$T_1 = \sum_{j=1}^{m} t_{ij} \geqslant \sum_{j=1}^{m} t_{2j} = T_{2j}$$

The problem as before is to find a path that satisfies the following conditions:

1. The path consists of straight-line segments, is continuous, and belongs completely to the rectangle $(0 \leq x \leq T_1 , 0 \leq y \leq T_2)$.
2. The path joins origin to the point P (T_1, T_2).
3. All segments of the path must be either horizontal, vertical, or at a 45-degree angle (up and to the right).
4. The path does not pass through the following domains (interiors of rectangles)

$$\sum_{p=0}^{u-1} t_1 \ , \ r_p < x < \sum_{p=0}^{u} t_1 \ , \ r_p \ ;$$

$$\sum_{q=0}^{v-1} t_2 \ e_q < y < \sum_{q=0}^{v} t_2 \ , \ e_q$$

where for each u, v = 1, 2, . . ., m such that

$r_u = e_v$, and $t_1 r_0 = t_2$, $e_0 = 0$.

5. Σ(vertical segment) + Σ(horizontal segments)

$$+ \frac{1}{\sqrt{2}} \ \Sigma \text{(diagonal segments) is minimum.}$$

An optimal feasible path satisfies all conditions and a feasible path satisfies conditions 1 through 4.

Let us consider the same example 2 jobs on 4 machines (A, B, C, D) with the following operating sequence. Job 1 is A_3 B_2 C_4 D_5 and job 2; C_2 B_5 A_3 D_2, where subscripts denote processing times. According to the feasibility theorem program, a $\bar{b}$ $\bar{c}$ d is feasible. The domain of all the lines corresponding to the program a $\bar{b}$ $\bar{c}$ d is given in Fig. 2.2, (interior represented by dotted lines). The boundary of domain of all feasible paths corresponding to above program consists of two lines ℓ and $\bar{\ell}$ (see Fig. 2.3), where

(a) ℓ does not run below $\bar{\ell}$;

(b) ℓ and $\bar{\ell}$ satisfy conditions 1 to 4. It is possible that ℓ and $\bar{\ell}$ have common segments (points).

The following procedure may be used to find optimal path for given feasible program.

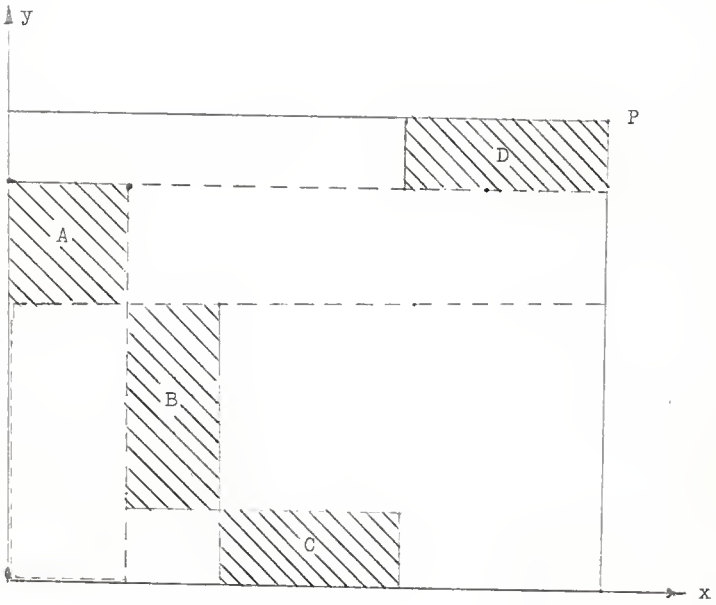

Fig. 2.2.

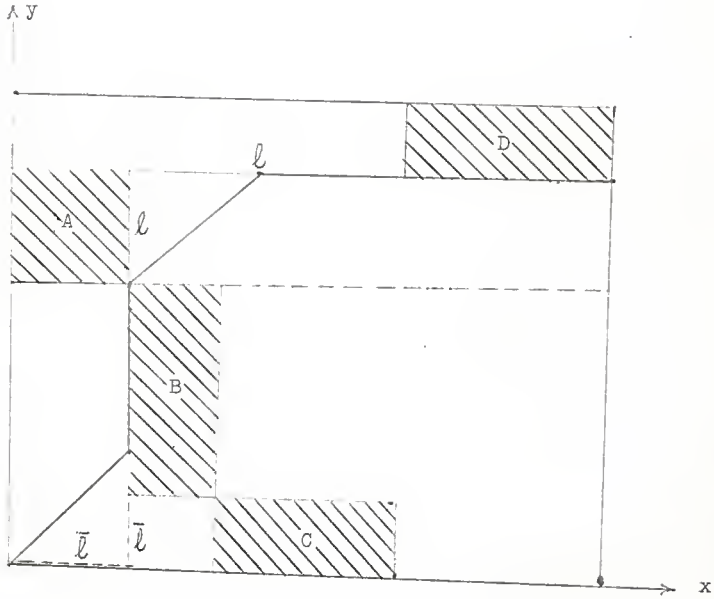

Fig. 2.3.

Step 1. Start from the origin in the 45-degree angle direction (to the right) until arriving at ℓ or $\overline{\ell}$. Then proceed to Step 2 (if moving in 45-degree angle direction, it is impossible to proceed to Step 2).

Step 2. Move along $\ell(\overline{\ell})$ to the right upwards until arriving at a node. (Origin, point P, and southeast or northwest corners of shaded rectangles are nodes.) Then go to Step 3.

Step 3. Start from the node in 45-degree angle direction (to the right) until arriving at ℓ or $\overline{\ell}$. Then go to Step 4.

Step 4. Repeat Steps 2 and 3 until point P is reached.

For the general case of the n x m problem, a method has been given which sometimes gives satisfactory results. The method does not guarantee that (a) the result obtained is a feasible solution, and (b) the result is optimal in case it is feasible. It consists in solving $\binom{n}{2}$ problems, each of which is 2 x m type. We then write the solution in the form of a program. From $\binom{n}{2}$ program of 2 x m type we form the n x m program.

Hardgrave and Nemhauser (31) have given another geometric model for the problem of scheduling n jobs on m machines so that the total time needed to complete the processing of all jobs is minimized. The most important assumptions made are:

1. There are no random or uncertain elements.
2. All the processing times are known.

3. Sequence of machines on which jobs are processed is specified.
4. A job may not be processed by more than one machine at a time.
5. A machine may not process more than one job at a time.
6. All machines are of different types.
7. Jobs must be processed without interruptions.

It is possible to remove assumptions 5 through 7 without difficulty. The approach is geometric interpretation of the sequencing problem in which feasible schedules are represented by paths in the n-dimensional rectangle. It is possible to construct a finite network in this rectangle such that every shortest path in the network corresponds to an optimal sequence. A simple and efficient algorithm for 2 x m is given. Although the algorithm is not limited by the number of machines, it loses efficiency rapidly with increase in number of jobs. For hand computations the number of jobs can be at the most three, but for efficient handling of the problem the number of jobs should be two. As shown before, feasible schedules can be represented geometrically within a closed rectangle (see Fig. 2.1). Rectangle ($0 \leq x \leq T_1$, $0 \leq y \leq T_2$) determines a region in which any point (t_1, t_2) represents a degree of completion for each job. Thus if S_{1m} and S_{2m} are the earliest start times for operations on machine m, the interior of the rectangle with lower left corner at (S_{1m}, S_{2m}) and upper right corner at ($S_{1m} + t_{1m}$, $S_{2m} + t_{2m}$) is an infeasible region. (See the shaded rectangles of Figs. 2.1, 2.2, and 2.3.) As before, paths

composed of straight-line segments from the origin (0, 0) to point $P(T_1, T_2)$ represent possible sequences. If we set the length of each possible path equal to its real processing time, i.e., a diagonal branch from (t_1, t_2) to $(t_1 + t, t_2 + t)$, a horizontal branch from (t_1, t_2) to $(t_1 + t, t_2)$, and a vertical branch from (t_1, t_2) to $(t_1, t_2 + t)$, all of them will be of length t. Thus:

$$\text{Length of a path} = \sum(\text{vertical segments}) + \sum(\text{horizontal segments}) + \frac{1}{\sqrt{2}} \sum(\text{diagonal movements}).$$

We will now proceed to show how a minimum path can be determined rigorously and efficiently by finding a shortest path through a finite network that will be called the schedule network.

The following rules will yield a sufficient number of nodes but, in general, a few more than are actually necessary to find a minimum path.

1. Starting at (0, 0), move diagonally until a region of infeasibility is encountered, say at (t_1, S_{2m}) (see Fig. 2.4). Note that this corresponds to intersecting an infeasible region on its bottom edge; if an infeasible region is intersected on its left side, the rules are similar.

2. Branch in the following two directions:

   (i) At (t_1, S_{2m}) move horizontally along the bottom edge of the rectangle until $(S_{1m} + t_{1m}, S_{2m})$, the point at which job 1 is completed on machine m.

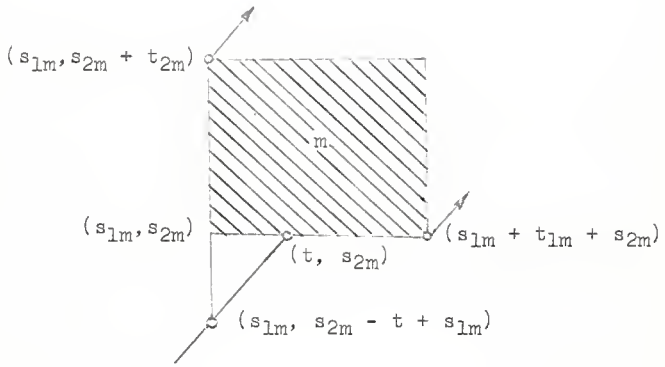

Fig. 2.4. An illustration of the branching rule (2).

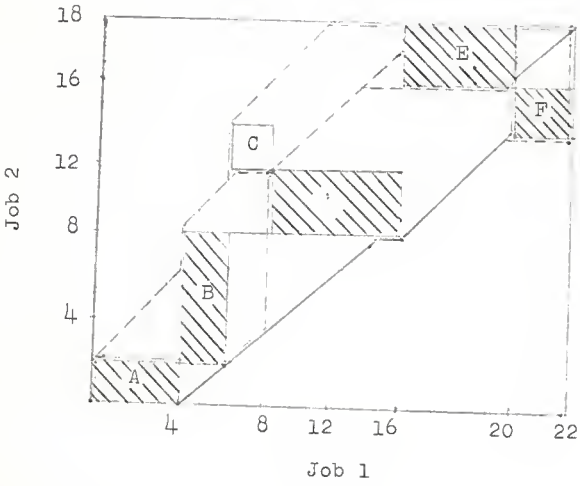

Fig. 2.5.

Return to Step 1, using $(S_{1m} + t_{1m}, S_{2m})$ as a new starting point.

(ii) At $(S_{1m}, S_{2m} - t_1 + S_{1m})$ move vertically along the left edge of the rectangle until $(S_{1m}, S_{2m} + t_{2m})$, the point at which job 2 is completed on machine m. Return to Step 1 using $(S_{1m}, S_{2m} + t_{2m})$ as a new starting point.

3. If the top or right edge of the outer rectangle is encountered, move along that edge to the finish point.

The procedure ends when all of the paths have reached the destination node $P(T_1, T_2)$. For the previous example the solution is obtained directly.

Let us consider an example with 2 jobs and 6 machines. Operations sequences are: Job 1, $A_4\ B_2\ C_2\ D_6\ E_5\ F_3$; and job 2, $A_2\ B_6\ D_3\ C_2\ F_2\ E_3$, where subscripts represent processing times. Figure 2.5 gives the solution for this example. The optimal path is given by the solid line; all other paths resulting from the application of the above rules are shown with dotted lines.

Starting at (0, 0) an infeasible region corresponding to machine A is hit immediately. One then moves horizontally to the point (4, 0) and vertically to (0, 2). From the point (4, 0) the next infeasible region, corresponding to machine D, is hit at the point (12, 8). From (12, 8) applying 2(i), there is a horizontal branch to (14, 8) and from 2(ii) there is a vertical branch from (8, 4) to (8, 11). All branches are then followed until they reach the terminal point P (22, 18). The path with minimum length corresponds to an optimal sequence.

The total time required is 24 time units. Only area F is below the line. Thus job 1 is processed before job 2 on all machines except F. The optimal program is a b c d e $\bar{f}$; other feesible programs are: a $\bar{b}$ c d e f, $\bar{a}$ $\bar{b}$ c d e f, end $\bar{e}$ b c d e $\bar{f}$.

2.2. The Generel Cese of n Jobs on m Mechines

This is en extension of the 2 x m problem. The principal result for the two-dimensionel case is true in the general cese also, namely, an optimal sequence corresponds to e shortest path from (0, . . ., 0) to (T_1, . . ., T_n) and the schedule network conteins such a shortest path. For n = 3, the general form of an infeasible region is shown in Fig. 2.6. The interior of the six-pronged cross end its intersections with the planes $S_i = 0$ and $S_i = T_i$ ere infeesible, [where S_i are the axes (x, y, z)] but other exterior surfeces are feesible.

The definition of the schedule network for the general case is similer to thet for the 2-job problem and can be informelly stated as follows:

1. Start et (0, 0, . . ., 0) and proceed diegonally until an infeesible region is encountered (which may, of course, be immedietely).
2. Whenever a block of en infeesible region is encountered, brench in one of the two possible weys eround it. These branches correspond to the two possible orders of processing the two jobs on the mechines that are represented by this infeasible region.
3. Along these brenches, proceed diagonally until enother block is hit. Then return to 2. Proceeding this way

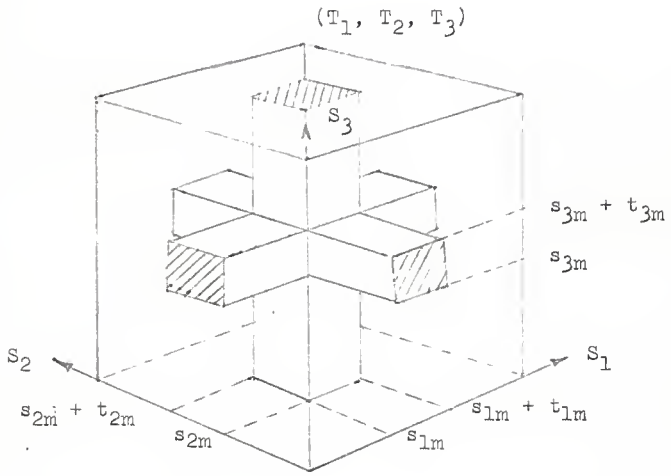

Fig. 2.6. A typical infeasible region for case n = 3.

we reach $(T_1, T_2, \ldots, T_n)$. It may happen that another infeasible block will be hit before it is possible to branch completely around the present block. In this case rule 2 is applied successively as many times as possible.

3.0. COMBINATORIAL APPROACH TO THE JOB-SHOP SCHEDULING PROBLEM

The main problem we are trying to solve is the following: There are n items, not all identical, which have to be processed through a number of machines of different types. The order in which the machines are to be used is not immaterial, since some of the processes must be carried out before others. Given the times required by i^{th} item on j^{th} machine, i = 1, 2, . . ., n; j = 1, 2, 3, . . ., m; determine the order in which the items should be fed into the machines so as to minimize the total time required to complete the processing of all the jobs. No job may be processed more than once on a given machine.

Mathematically, the problem is one of arrangements, which can be solved for any particular case by enumeration. However, a quick count of possible arrangements will show that as soon as the number of items reaches ten, the enumerative technique becomes unwieldy. Especially in the general case, the first item can be put on a machine in n different ways (there being n different items). The second item can be chosen from the remaining n - 1 and the third from n - 2. Thus there are (n) (n - 1) (n - 2) . . . (1) = n! ways of processing items on one machine. Since there are m machines, the items can be processed in $(n!)^m$ different ways.

The process of finding one or more permutations of the integers 1 through n which optimizes some objective function (say minimizes total elapsed time) is known as the combinatorial

approach to the job-shop scheduling problem. According to the presently available techniques it appears that this is the best approach to most of the job-shop scheduling problems. But still we lack a practical algorithm for complex problems.

By the term algorithm we mean a formal set of logical rules for the computation of some desired numerical function, viz., completion time of a feasible schedule, minimum man-hours for the schedule, maximum probability of lateness, minimum cost, etc. In the following pages of this section we shall present many typical algorithms which represent the efforts made to solve the scheduling problem with a combinatorial approach.

3.1. The Case of n Jobs on One Machine (n x 1)

In the case of n jobs on a single machine, if the execution times for the jobs are fixed, sequencing is no problem. For some applications the execution time for each job on the machine may be a random variable with a known probability distribution. B. P. Banerjee (5) has developed a simple algorithm by using minimization of the maximum probability of lateness as the optimization criterion. Under two assumptions:

1. The facility has to be constantly in use until all the jobs are completed;
2. the sequence of n jobs, once started, will not be interrupted before completion.

If T_i is planned (or desired) completion time for the i^{th} job and $i = 1, \ldots, n$, $T_1 \leq T_2 \leq T_3 \ldots \leq T_n$ is an

optimal sequence. This decision rule, apart from its simplicity of application, has some very desirable properties. Some of them are:

1. The rule is distribution free and is applicable regardless of the nature of processing time distributions.
2. If at any time it is decided to break the sequence to inject a priority job, the rest of the sequence still remains optimum.
3. The optimization process can be dynamic. Any job arriving at any time can be inserted into the waiting line in the order of desired completion time and the criterion for optimality is maintained. It is thus possible to relax the second assumption.
4. It has, inherent in it, the property of attaching a higher priority to lagging jobs, which is desirable in many job-shop situations.

3.2. The Case of n Jobs on Two Machines (n x 2)

The problem of n jobs through two machines has been solved by Johnson (47), Bellman (7, 10), and Mitten (58).

3.2.1. Johnson's Procedure to Minimize Total Time to Process n Jobs on Two Machines.

Johnson took the discrete case of two machines and n jobs under the assumption that the jobs are kept in the same order for both machines. Let us call the first machine on which a job is processed machine I and the other machine II. The objective function is the accumulated idle time on machine II, this being possible since the order of jobs is to be maintained.

Let a_i, b_i represent the time required to process the i^{th} item on first and second machines, respectively. Let x_i be the inactive time on the second machine immediately before the i^{th} item is processed on the second machine. Then

$$I_{nx2} = \sum_{i=1}^{n} x_i = \max_{1 \leq u \leq n} \sum_{i=1}^{u} a_i - \sum_{i=1}^{u-1} b_i \qquad (3.2.1)$$

where I_{nx2} represents the total idle time on machine II; refer to Fig. 3.1.

Proof:

$$\begin{aligned} x_1 &= a_1 \\ x_2 &= \max (a_1 + a_2 - b_1 - x_1 , 0) \\ &= \max (a_2 - b_1 , 0) \end{aligned} \qquad (3.2.2)$$

Whence

$$x_1 + x_2 = \max (a_1 + a_2 - b_1 , a_1) \qquad (3.2.3)$$

$$= \max \left(\sum_{1}^{2} a_i - \sum_{0}^{1} b_i \right) , \left(\sum_{1}^{1} a_i - \sum_{0}^{0} b_i \right)$$

where $b_0 = 0$. Similarly,

$$x_3 = \max \left(\sum_{i=1}^{3} a_i - \sum_{i=1}^{2} b_i - \sum_{i=1}^{2} x_i , 0 \right) \qquad (3.2.4)$$

and

$$\begin{aligned} \sum_{i=1}^{3} x_i &= \max \left(\sum_{i=1}^{3} a_i - \sum_{i=1}^{2} b_i , \sum_{i=1}^{2} x_i \right) \\ &= \max \left(\sum_{i=1}^{3} a_i - \sum_{i=1}^{2} b_i , \sum_{i=1}^{2} a_i - b_1 , a_1 \right) \end{aligned} \qquad (3.2.5)$$

The remainder of the proof is inductive.

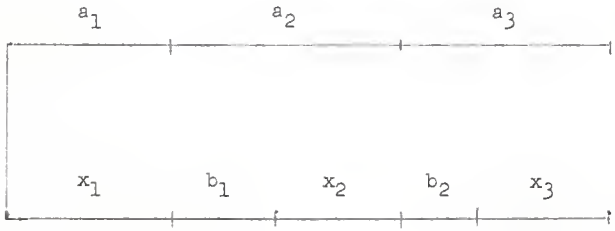
a_1
a_2
a_3
x_1
b_1
x_2
b_2
x_3

Fig. 3.1.

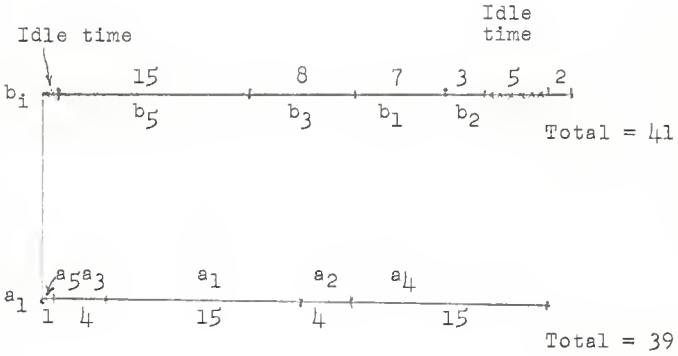
Idle time
Idle time
b_i
15
8
7
3
5
2
b_5
b_3
b_1
b_2
Total = 41
a_1
a_5
a_3
a_1
a_2
a_4
1
4
15
4
15
Total = 39

Fig. 3.2.

According to Johnson, an optimal ordering is determined by the rule: Item i precedes item j if

$$\min (a_i, b_j) < \min (a_j, b_i) \qquad (3.2.6)$$

If there is equality, either order is optimal provided that it is consistent with all definite preferences.

For a three-stage problem, the corresponding formula for the total idle time on the third machine, also derived by Johnson, is:

$$I_{nx3} = \max_{1 \leq u \leq v \leq n} \left[\sum_{i=1}^{u} a_i - \sum_{i=1}^{u-1} b_i + \sum_{i=1}^{v} b_i - \sum_{i=1}^{v-1} c_i \right] \qquad (3.2.7)$$

where a_i, b_i, c_i denote respectively the times required by the i^{th} item on the first, second, and third machines.

Example. Let us now illustrate the way in which this criterion (equation 3.2.6) may be applied. We follow the steps given below:

1. List a_i and b_i in two vertical columns

i	a_i	b_i
1	a_1	b_1
2	a_2	b_2
.	.	.
.	.	.
.	.	.
n	a_n	b_n

2. Determine the minimum of all the a_i and b_i.

3. If it is an a_i, place the corresponding item first.

4. If it is a b_i, place the corresponding item last.
5. Cross off times for that item.
6. Repeat the steps on reduced set of (n - 1) items.
7. In case of ties, order the items with smallest subscript first, for the sake of definiteness. If a tie between a_i and b_i, order the item according to third step.

To illustrate the method, consider this example. The rule yields (5, 3, 1, 2, 4) as minimal order with a total time of 41 units, and 6 units of idle time.

i	a_i	b_i
1	15	7
2	4	3
3	4	8
4	15	2
5	1	15

3.2.2. Bellman's Continuous Version of the Discrete Problem. Bellman (7, 10) has tackled the simplified problem of determining the optimal order when there are a large number of items of only a few different types. Even here, the original problem seems difficult to resolve. We shall use a device which works uniformly well throughout the theory of dynamic programming, namely, the replacement of a discrete problem by a continuous version. This continuous version may be solved with great ease.

Let us consider our problem under the following assumptions:

1. There are two machines and two different types of items.
2. The total number of items is large when compared to the times required to process any individual item.

Here in place of the expression

$$\sum_{i=1}^{u} a_i - \sum_{i=1}^{u-1} b_i ,$$

we consider the integral

$$I(u) = \int_0^u \big(a(t) = b(t)\big)\, dt \qquad (3.2.8)$$

The analogue of an arrangement of items is a characteristic function [the characteristic function $\phi(x)$ for a subset E with respect to a set F has the value 1 for points x in E, the value zero for points x in (F - E)] defined over the interval (0, T). This function determines a(t) and b(t) in the following way:

$$\begin{aligned} a(t) &= a_1\phi + a_2(1 - \phi) \\ b(t) &= b_1\phi + b_2(1 - \phi) \end{aligned} \qquad (3.2.9)$$

where (a_1, b_1) and (a_2, b_2) represent the times required on the first and second machines for first and second types, respectively. The function $\phi(t)$ is the characteristic function of the set over which the first item is processed. And $1 - \phi$ is the characteristic function of the set over which the second item is processed. The constraints upon ϕ are that it takes on

only the values 0 and 1, and in addition satisfies

$$\int_0^T \phi dt = k, \text{ (where k is an integer)} \qquad (3.2.10)$$

which is equivalent to the statement that k of the T items belong to the first type. If we set

$$\begin{aligned} \alpha &= (a_1 - a_2 + b_2 - b_1) \\ \beta &= (a_2 - b_2) \end{aligned} \qquad (3.2.11)$$

the problem is that of determining the quantity

$$I = \min_{\phi} \max_{0 \leq u \leq T} \left[\alpha \int_0^u \phi dt + \beta \right]$$

and determining the corresponding function ϕ.

A minimizing $\phi(t)$ (a solution) is given as follows:

$$\begin{aligned} (\alpha > 0) \qquad \phi(t) &= 1 \qquad (T - k \leq t \leq T) \\ &= 0 \qquad (0 \leq t < T - k) \\ (\alpha < 0) \qquad \phi(t) &= 1 \qquad (0 \leq t \leq k) \\ &= 0 \qquad (k < t \leq T - k) \\ (\alpha = 0) \qquad \phi(t) &\text{ is arbitrary} \end{aligned} \qquad (3.2.13)$$

Thus we see that two-stage process may be attacked by the functional equation approach of the theory of dynamic programming, and resolved without the use of an explicit formula for the idle time. This method is important since it is not always possible to obtain a tractable explicit analytic representation of the quantity that is to be minimized in many analogous problems.

3.2.3. n Jobs on Two Machines with Overlap (Use of Time Lags)

L. G. Mitten (58, 59) has given an analytical solution based upon two machines and n jobs with arbitrary start and stop lags and a common sequence. Bellman as well as Johnson, treated rather severely restricted version of the problem involving n jobs and two machines. Mitten has treated a somewhat less restrictive case which is applicable to a wider variety of problems; the function to be minimized is the same, i.e., the total idle time on machine II. The use of start and stop lags permits one to treat a variety of practical problems which previous models were unable to handle. A start lag is the minimum time which must elapse between starting a job on machine I and starting on machine II, and a stop lag is the minimum time which must elapse between completing a job on machine I and completing it on machine II.

Mitten has considered the following problem:

Two machines (I and II) process n jobs, with jobs to be run in the sequence S = (1, 2, . . ., n). Let a_i and b_i be (respectively) the times required by job i on machine I and machine II. Associated with each job i is a "start-lag" A_i ($\geqslant 0$) and a stop lag $B_i \geqslant 0$ (defined later). The following assumptions are made:

1. Each job is to be run first on machine I and then on machine II, using the same sequence (S) on both mschines.
2. On machine I, the jobs are to be run in the sequence S without interruption.

3. On machine II, job i is started as soon as possible after the completion of job i-1 on machine II, subject to the provisions that job i may not be started on machine II less than A_i time units after it was started on machine I, and job i may not be completed on machine II less than B_i time units after its completion on machine I.

By a proper choice of the start and stop lags, we can take care of overlapping production (in some situations it is possible to start the production of a given job on the second machine (II) before the completion of processing of the job on machine I; this is known as overlapping production), transport time between machines, and scheduling bottleneck machines (in most situations a small number of machines cause a vast majority of these problems; these highly loaded machines are referred to as "bottleneck" machines).

Minimizing the idle time on machine II is equivalent to minimizing the total time to complete all of the jobs on both machines (since we have a common sequence). Let x_i be the idle time immediately preceding the start of the job i on machine II. We shall use

$$I = \sum_{i=1}^{n} x_i \quad (= \text{total idle time on machine II})$$

as our objective function to be minimized.

The procedure for obtaining an optimal schedule, given by Mitten, is best explained with the aid of a simple numerical example. In Table 1 the entries in the first five columns

Table 1. Example of two-machine Gantt chart scheduling problem.

Given data					6=2-3	7=4-5	8=max(6,7)	9=8+3	10=8+5	11
1	2	3	4	5						
i	A_i	a_i	B_i	b_i	A_i-a_i	B_i-b_i	m_i	m_i+a_i	m_i+b_i	Running order
1	2	2	4	3	0	1	1	3	X	1
2	3	4	1	2	-1	-1	-1	X	1	5
3	4	6	3	2	-2	1	1	X	3	3
4	4	7	2	2	-3	0	0	X	2	4
5	3	4	4	3	-1	1	1	X	4	2

represent the given data. To find the optimal sequence, the following steps are carried out in the table.

1. In each row, subtract the value in column 3 from the value in column 2 and enter the result in column 6.
2. In each row, subtract the value in column 5 from the value in column 4 and enter the result in column 7.
3. In each row, take the (algebraically) larger value from column 6 and 7 and enter in 8.
4. In each row, put an X in column 10 if the value in the column 3 is less than the value in column 5; otherwise (if a_i b_i), put an X in column 9.
5. In each row with an X in the column 10, add the value in column 3 to the value in column 8 and enter the sum in the column 9.
6. In each row with an X in column 9, add value in column 5 to the value in column 8 and enter the sum in column 10.

7. The job (row) with the smallest value in column 9 is run first; job with next smallest value in column 9 is run second, and so on. The job (row) with smallest value in column 10 is run last; the job with next smallest value in column 10 is run next to last, etc. This running order is entered in column 11.

The sequence in column 11 leads to Fig. 3.3. The schedule requires 25 units of time to complete all five jobs on both machines.

3.3. The General Case--n Jobs on m Machines (n x m)

3.3.1. The Use of Branch and Bound Technique of Little, Murty, Sweeny, and Karel.

The Branch and Bound Technique originally developed by Little, Murty, Sweeny, and Karel (54) for solving the traveling salesman problem has been applied to some flow-shop scheduling problems. In order to use branch and bound technique given by Ignal and Schrage (40), one must be able to describe the problem as a tree, in which each node represents a partial solution. In addition, one must be able to write at each node a lower bound on the objective function (mean completion time) for all the nodes that emanate from it.

If there are n jobs in a jobset, the problem is to find a permutation or a sequence of the integers 1, 2, . . ., n, under the assumption that the job sequence is the same for both of the machines. The first node in the related tree structure corresponds to not having committed any one integer (i.e., jobs) to any position in the sequence. From this node there are n

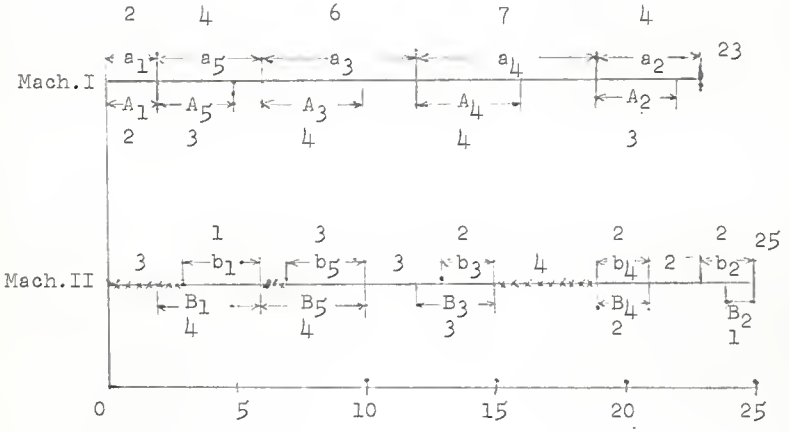
2
4
6
7
4
Mach.I
a_1
a_5
a_3
a_4
a_2
23
A_1
A_5
A_3
A_4
A_2
2
3
4
4
3
1
3
2
2
2
25
Mach.II
3
b_1
b_5
3
b_3
4
b_4
2
b_2
B_1
B_5
B_3
B_4
B_2
4
4
3
2
1
0
5
10
15
20
25

Fig. 3.3.

branches corresponding to the n possible integers (jobs) that can be assigned to first position in the sequence. From each of these nodes there are n - 1 branches corresponding to the n - 1 integers available to be placed in the second position, etc. Thus one can see that there are n! possible sequence or permutations, and $1 + n + n(n - 1) + \ldots + n!$ nodes in the entire tree.

The problem is to define an objective function and then to determine a lower bound for it. We will consider a 3-machine problem. The total idle time on third machine (III) is to be minimized, and since the order of the jobs on all machines is the same, it minimizes the make span. Each node represents a sequence of from 1 to n jobs. Let S_r be a sequence of a particular subset containing r jobs out of n. Let node P correspond to the sequence S_r. Let Time I (S_r), Time II (S_r), and Time III (S_r) be the times at which machines I, II, and III, respectively, complete processing on the last of the r jobs in the sequence. A lower bound on the make span of all the schedules that begin with the sequence S_r is given by

$$LB\ (S_r) = \max \left[\begin{array}{l} \text{Time I } (S_r) + \sum\limits_{i \in \bar{S}_r} a_i + \min\limits_{i \in \bar{S}_r} (b_i + c_i) \\ \text{Time II } (S_r) + \sum\limits_{i \in \bar{S}_r} b_i + \min\limits_{i \in \bar{S}_r} c_i \\ \text{Time III } (S_r) + \sum\limits_{1 \in \bar{S}_r} c_i \end{array} \right]$$

where a_i, b_i, c_i are processing times of the i^{th} job on machines

I, II, and III, respectively, and $\overline{S}_r$ is the set of n - r jobs that have not been assigned a position in the sequence S_r. $LB(P) = LB(S_r)$ is a lower bound on the make span for any node that emanates from node P, since all such nodes represent sequences from r + 1 to n jobs that begin with sequence S_r.

The technique, used by Ignall and Schrage, along with an example, is given below.

Make a list of nodes ranked by lower bound such that the node with the smallest lower bound is first. Make a list of attributes (jobs) that are in sequence in order Time I, Time II, Time III, for each node. Begin by having on the list only the node that has scheduled none of the jobs. Follow the steps given below to update the list recursively, until an optimal solution is reached.

1. Remove the first node from the list.
2. Create a new node for every job that the node just removed has not scheduled. Do this by attaching the unscheduled job to the end of the sequence of scheduled jobs.
3. Compute the lower bounds and the other attributes for these newly created nodes and insert them in the ranked list.
4. Go to 1 if no node has scheduled all n jobs on the list; otherwise, the problem is solved.

An example: consider the following 4-job 3-machine problem when the objective is minimizing makespan.

i	a_i	b_i	c_i
1	13	3	12
2	7	12	16
3	26	9	7
4	2	6	1

The tree and the list that are obtained by solving it by the above technique are shown in Fig. 3.4 and Table 2.

The lower bounds are written next to the nodes on the tree. In case of a tie, i.e., if a newly created node and another node had the same lower bound, the newly created node was put before the old node on the list. Note that node 231's lower bound, which is the makespan for sequence 2314, is less than or equal to the lower bound of any other 'unbranched form' node in the tree (or on the list). For example, it would be impossible for a sequence beginning with job 4 to have makespan < 63 and $62 \leq 63$, so there is no need to explore any sequences that begin with job 4. The same is true of sequences that begin with job 3, and the same applies to the other nodes, so 231 is optimal.

This technique is at least competitive with other methods (Wagner (58, 59), Dudek and Teuton (18)). It is particularly suited for problems of small size. As Little, et al., have mentioned, the Branch and Bound technique is a method of wide applicability. In the 2-machine problem, a variety of objective functions can be handled. For example, the lower bound for a

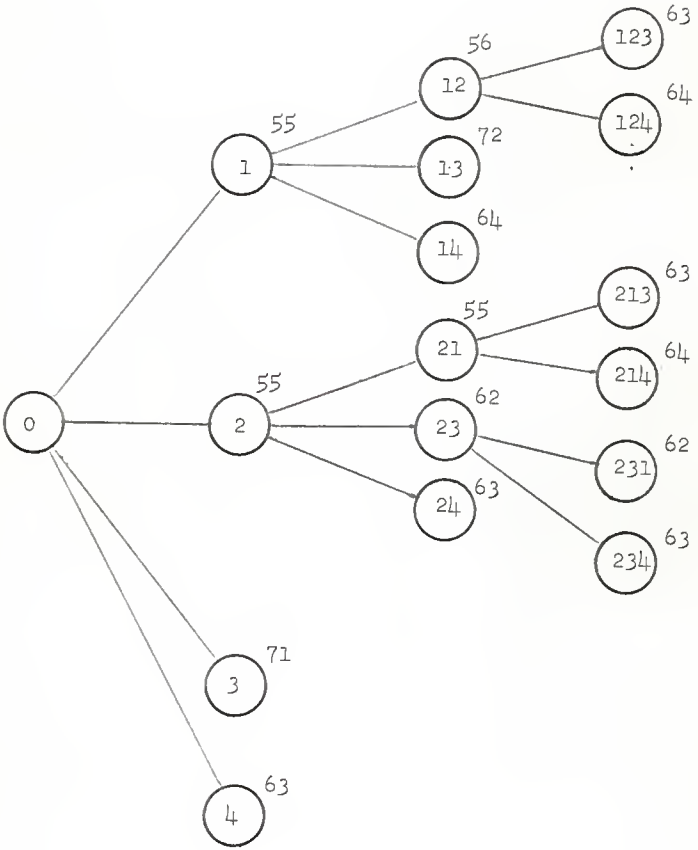
0
1
55
2
55
3
71
4
63
12
56
13
72
14
64
21
55
23
62
24
63
123
63
124
64
213
63
214
64
231
62
234
63

Fig. 3.4.

Table 2.

Node	Lower bound	Time II	Time III	Disposition
0	-	-	-	Branched form
1	55	16	28	Branched form
2	55	19	35	Branched form
3	71	35	42	
4	63	8	9	
21	55	23	43	Branched form
23	62	42	49	Branched form
24	63	25	36	
213	63	61	63	
214	64	57	64	
12	56	36	48	Branched form
13	72	48	55	
14	64	22	29	
123	62	61	63	
124	64	57	64	
231	62	60	62	An optimal sequence
234	63	51	63	

weighted sum of completion times or mean one-sided lateness or other criteria could be constructed, and the branch and bound technique applied.

Johnson and Bellman have given extensions of their (n x 2) problems to approximate 3-stage solutions for 3 machines and n jobs.

3.3.2. The Nonnumerical Approach of Akers and Friedman. In 1954, Akers and Friedman (2) developed a nonnumerical approach for eliminating nonfeasible and possibly not optimal programs out of $(n!)^m$ possible programs where n is the number of jobs and m is the number of machines. Machines are used for each job in a specified order. If a program does not meet this requirement, it is called nonfeasible. There are some technologically nonfeasible programs. For example, machining has to be done before painting or drilling before threading. The elimination of such programs is based purely on logical considerations without the recourse of any numerical data.

Later Akers (3) developed a graphical approach to this (2 x m) problem which offers very quick solution for the problem of 2 jobs on m machines. In this case the graph is in the plane of the paper but in the general case of n jobs on m machines, the graph would be in n-dimensional space and the solution becomes obscure. This nonnumerical technique is very useful for a smaller number of machines and where machine times are apt to change. The possibly optimal solutions can be kept ready and by applying new machine times an optimal solution can be found easily.

Bellman's continuous version of the discrete problem can be used to give approximate solution to the n x m problem. He has given a general formulation which can be tailored to particular situations. For example, where there are interchangeable machines, operators trained to work on all of the machines, and so on.

3.3.3. The Method of Active Feasible Schedules by Giffler and Thompson. Giffler and Thompson (25) have developed algorithms for solving problems to minimize the length of a production schedule. They have considered the following general assumptions:

1. Each of the n commodities must be processed by one or more of the m facilities (or machines).
2. The operation once started may not be interrupted.
3. A subset of the facilities may be equivalent (i.e., more than one machine of a given type).
4. Some pairs of operations may be performed in succession. Some pairs must be performed in succession.
5. The time needed to perform each operation is given and also minimum delay after the start of first operation and before the second operation may be started in case of pairs of operation, is given.
6. For convenience we assume operations for a commodity are performed in linear sequence, i.e., no commodity is processed more than once by any facility or pair of equivalent facilities.

7. The time to perform operations is independent of the order in which they are performed.

The algorithms generate one or all the schedules of a particular subset of all the possible $(n!)^m$ called the active feasible schedules. An active feasible schedule is defined as a schedule with the following properties: (a) no machine is idle for a length of time sufficient to completely process a simultaneously idle commodity; and (b) whenever an assignment of a commodity to a machine has been made, its processing is started at the earliest time that both machine and commodity are free.

In these algorithms a subset of facilities may be equivalent and some pairs of operations may be performed in succession. The completion date problems can be easily handled if after discarding those active schedules in which due dates are not met, we are left with one or more feasible active schedules. Their main weakness is that we cannot handle cases where the criteria for optimality is other than the shortest schedule. Thus one could require of an optimal schedule that it minimize the total idle time of all facilities and/or all commodities. We could also require that the total dollar value of the in-process inventory be minimized.

3.3.4. An Approximate Method by Dudek and Teuton when no Passing is Allowed. Dudek and Teuton (18) have given an algorithm that will determine an optimum sequence for n jobs processed through m machines when no passing is allowed, that is, the order in which n jobs pass through each of the machines be identical. The algorithm does not generate all the optimal

sequences.

Statement of the problem: Given n jobs to be processed on m machines (m 3), I, II, III, . . ., m in the order I, II, III, . . ., m, determine a sequence (one or more) that will minimize total elapsed time. The algorithm is applicable under the following assumptions:

1. No machine may process more than one job at any given time.
2. A job once started must be completed.
3. Processing time for each operation is known and finite. It is independent of the order of operations.
4. There are no due dates for any job.
5. The jobs are processed by machines as soon as possible and in a common order.
6. Transport time between machines may be considered negligible or as part of the processing time on the preceding machine.
7. Storage space for partially finished jobs is available.

We see that almost all of the assumptions (except 4 and 5) are in line with most of the practical problems. In most of the cases, especially where there is less automation, the processing times are variable but good approximations can be made by using average times.

In the mathematical formulation, total idle time accumulated on the last machine to process each job is minimized. This in turn minimizes the total elapsed time required for processing of all the jobs because of the assumption that all the

jobs are processed through all the machines in a common order. The analysis of idle time follows Johnson's approach to two-stage and three-stage sequencing problems. The development proceeds from three-stage case to a four-stage case and then to the general m stage case.

Let a_i = time required by job i on machine I

b_i = time required by job i on machine II

c_i = time required by job i on machine III

Ib_i = idle time on machine II from the end of job i - 1 to the start of job i

I_{c_i} = idle time on machine III from the end of the job i - 1 to the start of job i

T_e = total elapsed time for jobs 1, 2, . . ., n.

Now the problem is to find one or more permutations of the integers 1 through n which will minimize T_e. An example of a possible sequence for n = 4 can be represented by the Gantt Chart in Fig. 3.5. For the three-stage case

$$T_e = \sum_{i=1}^{4} c_i + \sum_{i=1}^{4} I_{c_i}$$

so the problem becomes one of minimizing $\sum_{i=1}^{4} I_{c_i}$.

To determine if job j should precede job j + 1, we start with sequence S' and from it obtain another sequence S" by interchanging the j^{th} and (j + 1)st jobs.

S' = 1, 2, 3, . . ., j - 1, j, j + 1, j + 2, . . ., n

S" = 1, 2, 3, . . ., j - 1, j + 1, j, j + 2, . . ., n.

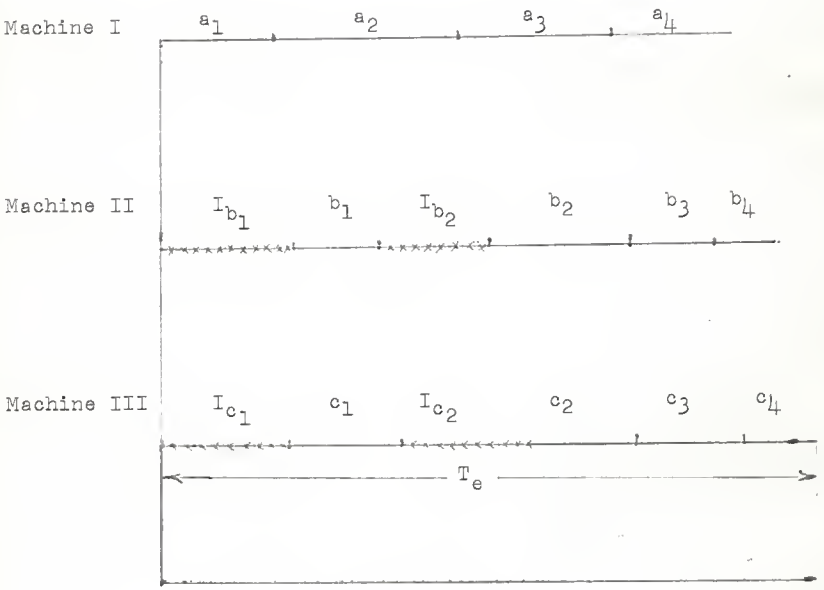

Fig. 3.5. Gantt chart (3-machine).

Let $K_v\ (III) = \sum_{i=1}^{i=v} b_i - \sum_{i=1}^{v-1} c_i + \max_{1 \leq u \leq v} (\sum_{i=1}^{u} a_i - \sum_{i=1}^{u-1} b_i)$

and

$$Q\ (II) = \max_{1 \leq u \leq j-1} (\sum_{i=1}^{u} a_i - \sum_{i=1}^{u-1} b_i)$$

and $K_v(III)'$ represent the value of $K_v(III)$ for sequence S', and $K_v(III)''$ represent the value of $K_v(III)$ for the sequence S". The following (m - 1) conditions have to be satisfied before a definite decision can be made on the preference of the two sequences, S' and S". Thus for an m-stage process, job j should precede job j + 1 if

(Condition 1)

$$\max \left[K_j(m)', K_{j+1}(m)' \right] < \max \left[K_j(m)'', K_{j+1}(m)'' \right]$$

(Condition 2)

$$\max \left[Q(m-1), K_j(m-1)', K_{j+1}(m-1)' \right] \leq \max \left[Q(m-1), K_j(m-1)'', K_{j+1}(m-1)'' \right]$$

. . . . . .

. . . . . .

(Condition m-1)

$$\max \left[Q(II), K_j(II)', K_{j+1}(II)' \right] \leq \max \left[Q(II), K_j(II)'', K_{j+1}(II)'' \right]$$

This m-stage rule is valid only if (m-1) conditions hold. If condition 1 holds, but one (or more) of the remaining conditions is not satisfied, then a decision cannot be made on the

preferability of S' or S". The algorithm given below lists the necessary steps to go through when this situation arises.

A scheduled job is defined as one that has been selected by the algorithm as the 1st, 2nd, 3rd, etc., job of the feasible schedule (one generated by algorithm). An unscheduled job is defined as one that has not been selected by the algorithm to fill a specific position of a feasible sequence.

The Algorithm. It is assumed that (j - 1) jobs have been scheduled for the feasible sequence under consideration.

Step 1. List the (j - 1) scheduled jobs in their scheduled sequenced position of the sequence table as illustrated in Table 3.

Step 2. Determine the min $\left[a_i + b_i + c_i \ldots (m-1)_i \right]$ for all remaining unscheduled jobs. In case of a tie, select one with max m_i. Place the corresponding job and its processing times in the j^{th} sequence position of the sequence table.

Step 3. Place one of the remaining (n - j) unscheduled jobs in the (j + 1)st sequence position of the sequence table.

Step 4. See if condition 1 is satisfied.

Step 5. Apply one of the following:

(a) If Condition 1 is satisfied, repeat steps 3 to 5 for each remaining possible sequence, that is, the reduced set of (n - 1 - j), (n - 2 - j), etc., remaining unscheduled jobs; continue to step 6.

Table 3. Sequence table.

Sequence position	Job i	a_i	b_i	c_i	. . .	m_i
1	4	a_4	b_4	c_4		m_4
	3	a_3	b_3	c_3		m_3
. . . . .						
. . . . .						
(j - 1)	1	a_1	b_1	c_1		m_1
(j)						
(j + 1)						
.						
.						
.						
n						

(b) If Condition 1 is not satisfied because of an equality, repeat step 3 through 5 for each remaining possible sequence; continue to step 6.

(c) If condition 1 is not otherwise satisfied, replace the job currently in sequence position (j + 1), repeat steps 3 to 5 for each remaining possible sequence, if all the unscheduled jobs have been tried in sequence position (j), go to step 6.

Step 6. If (a) the job presently in the j^{th} position satisfies Condition 1 for all remaining unscheduled jobs, go to step 7a.

(b) The job presently in the j^{th} position fails to satisfy Condition 1 because one or more of the

remaining unscheduled jobs yield an equality; go to step 7b.

(c) None of the remaining unscheduled jobs satisfies (a) or (b); go to step 7c.

Step 7. Apply one of the following:

(a) If 6a occurs, determine if condition 2 through m - 1 are satisfied for all remaining (n - j) unscheduled jobs.

(1) If conditions 2 through m - 1 are satisfied for all remaining unscheduled jobs, schedule the job in sequence position (j) as the next job of the feasible sequence.

(2) If conditions 2 through m - 1 are not satisfied for i remaining (n - j), unscheduled jobs, it is necessary to assume that the job presently in the j^{th} sequence position as well as the remaining i unscheduled jobs that do not satisfy this condition as the job of (i + 1) feasible sequences.

(b) If 6b occurs, it is necessary to assume the job in the j^{th} sequence position as well as k of the remaining unscheduled jobs as the j^{th} job of (k + 1) feasible sequences, where k is equal to the number of remaining unscheduled jobs that had yielded an equality in test of Condition 1.

(c) If 6c occurs, it is necessary to assume (turn by turn) all remaining (n + 1 - j) unscheduled jobs

as the j^{th} job of (n - 1 - j) feasible sequences. Go to step 8.

Step 8. After having placed one job in the j^{th} position of the sequence table, repeat step 1 through 7 until (n - 2) jobs have been sequenced into a feasible solution. (If more than one job were assumed in step 7, the first is assigned to position j and the others were put aside until step 11.)

Step 9. To determine the (n - 1)st job of a feasible solution, determine which of the two remaining jobs satisfies Condition 1. Place the respective job in sequence position n. In case of a tie, select either as job (n - 1) of the feasible sequence.

Step 10. Enumerate the feasible sequence to determine total elapsed time.

Step 11. If more than one job were assumed for position j in step 7, select another (the first was chosen in step 8) and repeat the step 1 through 10 until the values put aside in 8 are used.

Step 12. Determine the sequence (one or more) that yield the minimum total elapsed time for processing times under consideration. This sequence becomes the optimal sequence as it minimizes the total elapsed time T.

The conditions 1 through m - 1 can be stated differently. For example, for three machines, job j should precede job (j + 1) if

(Condition 1)

$$\min \left[b_j - \max \left\{ Q(II),\ K_j(II)'' \right\},\ c_{j+1} - \max \left\{ Q(II),\ K_j(II)'',\ K_{j+1}(II)'' \right\} \right]$$

$$\min \left[b_{j+1} - \max \left\{ Q(II),\ K_j(II)' \right\},\ c_j - \max \left\{ Q(II),\ K_j(II),\ K_{j+1}(II)' \right\} \right]$$

and (Condition 2)

$$\max \left[Q(II),\ K_j(II)',\ K_{j+1}(II)' \right] \leq \max \left[Q(II),\ K_j(II)'',\ K_{j+1}(II)'' \right]$$

Now we will solve the previous example by using the above algorithm.

The data are given in tabular form. We will determine the order of jobs such that the total time to complete the processing of all jobs is minimized (see Table 4).

Table 4. Processing time (hours).

Job	Machine I	Machine II	Machine III
1	13	3	12
2	7	12	16
3	26	9	7
4	2	6	1

First we find $\min(a_i + b_i)$ is 8 hours for job 4. Thus we place job 4 in sequence position 1. Placing job 1 in sequence position 2 yields the following calculations (Table 5):

Table 5. Sequence table (job 4 vs. job 1).

Sequence position	Job i	a_i	b_i	c_i
1	4	2	6	1
2	1	13	3	12

$$\min\left[6 - \max(0, 13),\ 12 - \max\{0, 13, 12\}\right] \not< \min\left[3 - \max\{0, 2\},\ 1 - \max\{0, 2, 9\}\right]$$

or $\quad -7 \not< -8.$

So condition 1 is not satisfied and according to step 5c we place job 1 in sequence position 1. Comparing job 1 with job 2, and following the calculation for condition 1, yields Table 6.

Table 6. Sequence table (job 1 vs. job 2).

Sequence position	Job i	a_i	b_i	c_i
1	1	13	3	12
2	2	7	12	16

$b_j = 3 \qquad c_{j+1} = 16 \qquad b_{j+1} = 12 \qquad c_j = 12$

$Q(II) = 0 \qquad K_j(II)'' = 7 \qquad K_{j+1}(II)'' = 8$

$K_j(II)' = 13 \qquad K_{j+1}(II)' = 17$

$$\min\left[3 - \max(0, 7),\ 16 - \max(0, 7, 8)\right] \not< \min\left[12 - \max(0, 13),\ 12 - \max(0, 13, 17)\right]$$

or $\quad -4 \not< -5.$

Again, condition 1 is not satisfied, so according to 5c, place job 2 in sequence position 1 (see Table 7).

Table 7. Sequence table (job 2 vs. job 1)

Sequence position	Job i	a_i	b_i	c_i
1	2	7	12	16
2	1	13	3	12

$b_j = 12 \qquad c_{j+1} = 12 \qquad b_{j+1} = 3 \qquad c_j = 16$

$Q(II) = 0 \qquad K_j(II)'' = 13 \qquad K_{j+1}(II)'' = 17$

$K_j(II)' = 7 \qquad K_{j+1}(II)' = 8$

(Condition 1)

$$\min\left[12 - \max(0, 13), 12 - \max(0, 13, 17)\right] \not< \min\left[3 - \max(0, 7), 16 - \max(0, 7, 8)\right]$$

or $\quad -5 \not< -4$.

(Condition 2)

$$\max\left[(0, 7, 8)\right] \quad \max\left[(0, 13, 17)\right], \text{ or } (8 \leq 17).$$

Thus conditions 1 and 2 are satisfied for job 4 vs. job 1. Similarly, we see job 2 satisfies both conditions for all remaining unscheduled jobs. So we schedule job 2 first. Now we again scan various processing times to determine $\min(a_i + b_i)$. We eliminate job 2 from the comparison. We find job 4 has $\min(a_i + b_i)$. But as we have already seen, job 4 vs. job 1 does not satisfy condition 1, so we put job 1 in sequence position 2. We will see if job 1 satisfies conditions 1 and 2 when

placed in sequence position 2, placing job 3, one of the remaining (n - j) unscheduled jobs, in sequence position 3 (Table 8).

Table 8. Sequence table (job 1 vs. job 3).

Sequence position	Job i	a_i	b_i	c_i
1	2	7	12	16
2	1	13	3	12
3	3	26	9	7

$b_j = 3 \qquad c_{j+1} = 7 \qquad b_{j+1} = 9 \qquad c_j = 12$

$Q(II) = 7$

$K_j(II)'' = 7 + 26 - 12 = 21$

$K_{j+1}(II)'' = 46 - 12 - 9 = 25$

$K_j(II)' = 20 - 12 = 8$

$K_{j+1(II)}' = 46 - 15 = 31$

(Condition 1)

$$\min\left[3 - \max(7, 21),\ 7 - \max(7, 21, 25)\right] < \min\left[9 - \max(7, 8),\ 12 - \max(7, 8, 31)\right]$$

or $\quad -18 < -19.$

Thus condition 1 is not satisfied, so we replace (step 5c) job 1 by job 3 (Table 9).

Table 9. Sequence table (job 3 vs. job 1).

Sequence position	Job i	a_i	b_i	c_i
1	2	7	12	16
2	3	26	9	7
3	1	13	3	12

$b_j = 9 \qquad c_{j+1} = 12 \qquad b_{j+1} = 3 \qquad c_j = 7$

$Q(II) = 7 \qquad K_j(II)'' = 7 + 13 - 12 = 8$

$K_{j+1}(II)'' = 46 - 12 - 3 = 31$

$K_j(II)' = 33 - 12 = 21$

$K_{j+1}(II)' = 46 - 21 = 25$

(Condition 1)

$$\min\left[9 - \max(7, 8), 12 - \max(7, 8, 31)\right]$$

$$\min\left[3 - \max(7, 21), 7 - \max(7, 21, 25)\right]$$

or $\quad -19 < -18.$

(Condition 2)

$$\max(7, 21, 25) \leq \max(7, 8, 31)$$

or $\quad 25 \leq 31.$

Table 10. Sequence table (job 3 vs. job 4)

Sequence position	Job i	a_i	b_i	c_i
1	2	7	12	16
2	3	26	9	7
3	4	2	6	1

$b_j = 9$ $\quad c_{j+1} = 1$ $\quad b_{j+1} = 6$ $\quad c_j = 7$

$Q(II) = 7$ $\quad K_j(II)'' = 7 + 2 - 12 = -3$

$K_{j+1}(II)'' = 35 - 12 - 6 = 17$

$K_j(II)' = 33 - 22 = 11$

$K_{j+1}(II)' = 35 - 21 = 14$

(Condition 1)

$$\min\left[9 - \max(7, -3),\ 1 - \max(7, -3, 17)\right] \leq \min\left[6 - \max(7, 21),\ 7 - \max(7, 21, 14)\right]$$

or $\quad -16 \leq -15$.

(Condition 2)

$$\max(7, 21, 14) \leq \max(7, -3, 17)$$

or $\quad 14 \leq 17$.

Hence conditions 1 and 2 are satisfied by job 3 for remaining unscheduled jobs, so job 3 goes to sequence position 2.

2	3	

Now since two jobs are left we will determine the next job in the sequence according to step 9. Let us try job 1 if it satisfies condition 1 when placed next in sequence (Table 11).

Table 11. Sequence table.

Sequence position	Job i	a_i	b_i	c_i
1	2	7	12	16
2	3	26	9	7
3	1	13	3	12
4	4	2	6	1

$b_j = 3 \qquad c_{j+1} = 1 \qquad b_{j+1} = 6 \qquad c_j = 12$

$Q(II) = 21 \qquad K_j(II)'' = 33 + 2 - 21 = 14$

$K_{j+1}(II)'' = 48 - 21 - 6 = 21$

$K_j(II)' = 46 - 21 = 25$

$K_{j+1}(II)' = 48 - 24 = 24$

$$\min\left[3 - \max(21, 14),\ 1 - \max(21, 14, 21)\right] < \min\left[6 - \max(21, 25),\ 12 - \max(21, 25, 24)\right]$$

or $\quad -20 < -19$.

So condition 1 is satisfied. Hence this feasible sequence, 2314, is the optimal sequence, since there is only one feasible sequence.

To complete the problem let us find total elapsed time (Table 12).

Table 12.

Job	Machine I		Machine II		Machine III	
	Time in	Time out	Time in	Time out	Time in	Time out
2	0	7	7	19	19	35
3	7	33	33	42	42	49
1	33	46	46	49	49	61
4	46	48	49	54	61	62

Hence total time elapsed to complete processing all jobs is 62 hours.

Dudek and Teuton have asserted without proof that the general algorithm will generate one or more optimal sequences but it will not necessarily generate all the possible optimal sequences.

It is possible to treat the important jobs; the algorithm will function all right when the jobs have been selected to fill first K positions of the feasible solution.

The authors have shown that the number of feasible schedules generated increases when n, m, or both, increase but the amount of computation required to determine feasible solutions remains small as compared to that required for the complete enumeration.

Though W. Karush has shown with a 3 x 3 example that the algorithm fails to give an optimal sequence contrary to the assertion made by Dudek and Teuton, yet this remains one of the best available algorithms at present.

4.0. INTEGER LINEAR PROGRAMMING APPROACH TO JOB-SHOP SCHEDULING PROBLEM

The job-shop scheduling problem is essentially a problem of determining the order of jobs on different machines so that some criteria (e.g., length of time, man-hours, expected profit, expected cost, etc.) are optimized. It is assumed that a job cannot be scheduled to start until all of its immediate predecessors have been finished. Hence some device is needed to signal when each job is finished. The mechanism used for this is a variable that is equal to one if the given job is finished and equal to zero otherwise. Using this idea and variants of it, one can formulate the job-shop scheduling problem as an integer linear programming problem as shown by Manne (56). There have been at least three such formulations of the problem.

Many practical linear programming problems by nature demand integer valued solutions. Because activities and resources, such as machines and people, are frequently indivisible. Of course, such problems can be solved as ordinary L. P. problems, and attempts can then be made to "round" the answers obtained to give integer solutions. In doing this we are not guaranteed an optimal answer. Hence integer L. P. solutions to the problems are required.

Integer programming algorithms such as Gomory (29) differ from the ordinary L. P. algorithms in that new constraints are created and added to the tableau as the algorithms proceed.

4.1. Definition of the Problem

There are n commodities or jobs to be processed on m machines. We assume that the order in which the commodities are to be processed by the machines is completely specified. This can be done easily by means of a matrix called the facility (machine) sequence matrix; two examples are given in Fig. 4.1 and Fig. 4.2. The indices in the matrix of Fig. 4.1 are those of the facilities that are to operate on the given job and the order in which these indices appear is the order in which the job is to be "routed" to the various machines. It is assumed here that all the job times are 1. The indices of the matrix in Fig. 4.2 are again the indices of the machines in the order in which the job is to be processed. The superscripts of these indices are the corresponding job times.

Feasible schedules are most easily specified by means of Gantt Charts which are illustrated in Fig. 4.3. For example, a feasible schedule for the problem shown in Fig. 4.1 is given in Fig. 4.3. The entries in the chart are the indices of jobs, and these appear in the order in which each machine processes them. Columns here represent different time intervals. In any column no job appears more than once, meaning that only one job is processed at one machine at a time. Idle times are represented by dashes.

Similarly, a feasible schedule for problem given in Fig. 4.2 is given in Fig. 4.4. Here the repetition of an index or dash indicates the number of time periods that the job with that index or idle time are on the machine. This problem was

J_1	3	1	2	4	6
J_2	2	3	5	6	1
J_3	3	4	6	1	2
J_4	2	1	3	4	5
J_5	3	2	5	6	1
J_6	2	4	6	1	5

Fig. 4.1.

J_1	1^5	2^2	3^8	4^7
J_2	3^8	1^4	4^5	2^3
J_3	4^6	1^7		

Fig. 4.2.

M_1	-	-	4	1	3	6	2	5
M_2	6	4	2	5	1	3	-	-
M_3	5	3	1	2	4	-	-	-
M_4	-	6	3	-	-	1	4	-
M_5	-	-	-	-	2	5	6	4
M_6	-	-	6	3	-	2	5	1

Fig. 4.3.

formulated as in Integer L. P. by Wagner (83), Bowman (12), and Manne (56), which are discussed.

4.2. Wagner's Model with Two Tabular Arrays

Wagner considers the following machine scheduling or sequencing problem:

"Given n items, each to be processed on one or more of m machines, the order of processing for an item being partially or entirely specified, find the sequencing of items on the machines which minimizes the total elapsed time to complete the manufacture of all the items."

Assumptions:

1. Manufacturing time of an item on a machine is specified (i.e., non-stochastic).
2. No job is processed on any machine more than once.
3. In-process inventory is allowable, i.e., passing is allowed.

4.2.1. Formulation of the Model. We may picture the restrictions characterizing a specific scheduling problem by means of two tabular arrays of the type shown in Figs. 4.1 and 4.2.

In the first matrix, a row corresponds to one of the n items, and a column corresponds to a process stage in the manufacture of the item. For example, if item i must be processed on machine 1, 2, . . ., m in that order, we define for the i^{th} row the entry at process stage 1 to be "machine 1", at process stage 2 to be "machine 2", . . ., at process stage m to be "machine m".

If an item is not placed on every machine, then the number of process stages is less than m. A minor modification of the model may be made to allow for multiple processing. If at some point of the manufacturing process of an item, the item is to be placed on 'q' machines out of a subset (Q) of machines, the specific order of manufacturing being unimportant, then we define the entry at that process stage to be subset (Q) and the specification q, e.g., item i at the fifth stage may be required to be placed on any of three (= q) machines in the group (Q) = (machine 5, machine 10, machine 16, machine 17, machine 25). Note that all of the items do not necessarily have the same number of process stages. We shall consider those production situations in which it is possible to analyze the manufacturing process for each item as a consecutive sequence of stages each comprising processing on one or more machines, and such that the order of manufacturing within a process stage is irrelevant.

In the second matrix the rows correspond to the items and the columns, to the m machines. The entry $t_i^{(k)}$ at the intersection of i^{th} row and k^{th} column represents the manufacturing time for item i on machine k. If the first matrix indicates that under no circumstances is an item to be placed on a particular machine, then the corresponding element in the second matrix will not be defined. We postulate that our time units are such that every $t_i^{(k)}$ is an integer.

We will use the following nonnegative integer valued variables:

$$x_{ij}^{(k)} = \begin{cases} 1 & \text{if item i is scheduled in order-- position j on machine k.} \\ 0 & \text{if item i is not scheduled in order-- position j on machine k.} \end{cases}$$

$$h_j^{(k)} = \{\text{time at which the item scheduled in order position j begins processing on machine k}$$

$$s_j^{(k)} = \{\text{time elapsing on machine k between completion of the processing the item scheduled in order--position j and initiation of processing the item scheduled in order--position j + 1}$$

$$s_0^{(k)} = \{\text{time elapsing on machine k between the initiation of production and the start of the processing of the item scheduled in the first position.}$$

By "the item scheduled in order-position j . . . on machine k" we mean that, prior to this item being placed on machine k, (j - 1) items have previously been processed.

Let n(k) = maximum number of items that might be processed on machine k. It represents an upper bound to the number of order positions that need to be considered for machine k.

Let N(k) be set of items for which machine k appears at some stage. First we consider the constraints that insure that each item i completes necessary operation within every process stage. If for a stage p the item i must be placed on machine k, we require

$$\sum_{j=1}^{n(k)} x_{ij}^{(k)} = 1 \qquad (4.2.1)$$

That is, the item i must appear in some ordered position on machine k. The restriction that $x_{ij}^{(k)}$ be integer-valued implies that the item will be scheduled once, and only once, on machine k. If there are Q machines listed for process stage p and the item must be placed on every one of the Q machines, then we have a set of Q equations of the form of equation (4.2.1), one for each such machine. A similar statement holds for relations below.

If for a given item i and process stage p the item must be placed on one machine out of a group of machines k_1, k_2, . . ., k_Q, we need

$$\sum_{j=1}^{n(k_1)} x_{ij}^{(k_1)} + \sum_{j=1}^{n(k_2)} x_{ij}^{(k_2)} + \cdots + \sum_{j=1}^{n(k_Q)} x_{ij}^{(k_Q)} = 1 \qquad (4.2.2)$$

Finally, if for a process stage p the item i must be placed on all of q machines out of a group of machines k_1, k_2, . . ., k_Q, we require

$$\sum_{j=1}^{n(k_1)} x_{ij}^{(k_1)} = 1 - \delta_1 \qquad (4.2.2a)$$

$$\sum_{j=1}^{n(k_2)} x_{ij}^{(k_2)} = 1 - \delta_2 \qquad (4.2.2b)$$

$$\vdots$$

$$\sum_{j=1}^{n(k_Q)} x_{ij}^{(k_2)} = 1 - \delta_Q \qquad (4.2.2Q)$$

$$\delta_1 + \delta_2 + \ldots + \delta_Q = Q - q \ldots \qquad (4.2.3)$$

where δ_k is restricted to be 0 or 1.

Equations (4.2.2) and (4.2.3) and the integer restrictions on δ insure that item i is processed on exactly q out of Q machines.

The second set of constraints guarantees that no more than one item be assigned the j^{th} ordered position on a machine. For each machine k, we require

$$\sum_{i \in N(k)} x_{ij}^{(k)} \leq 1 \qquad j = 1, 2, \ldots, n(k) \quad (4.2.4)$$

The above given constraints do not guarantee, firstly, an item be scheduled for process stage p and completed before it is scheduled for process stage p + 1, or even that, secondly, an item not to be processed on two or more machines during the same time. Consequently we need a set of (linear) relations implying that the production schedule observes the process-stage restrictions and does not call for the item being placed on more than one machine at one time. Here we use a "shorthand" notation and to give explicit relations for $h_j^{(k)}$.

Let

$$T x_j^{(k)} = \sum_{i \in N(k)} t_i^{(k)} x_{ij}^{(k)} \qquad j + 1, 2, \ldots, n(k) \quad (4.2.5)$$

Given equation (4.2.4), $Tx_j^{(k)}$ represents processing time of the j^{th} ordered item on machine k. Then for each machine k it may be verified that

$$h_1^{(k)} = s_0^{(k)} \tag{4.2.6a}$$

$$h_j^{(k)} = T\, x_1^{(k)} + T\, x_2^{(k)} + \ldots + T\, x_{j-1}^{(k)} + s_0^{(k)} + s_1^{(k)} + \ldots + s_{j-1}^{(k)} \qquad j = 2, 3, \ldots, n(k) \tag{4.2.6b}$$

Equation (4.2.6b) states that the item in order-position j on machine k commences processing at a time equal to the sum of the manufacturing and idle periods accumulated from initial commencement of production.

Now we consider restrictions. First restriction that for a particular item i any machining taking place in process stage p + 1 may commence only after all machining is completed in process stage p. We suppose in i^{th} row of matrix (first) machine k_1 appears in the entry column p and machine k_2 appears in the entry for column p + 1. Suppose further that $x_{ij'}^{(k_1)} = 1$ and $x_{ij''}^{(k_2)} = 1$, i.e., item i is scheduled in order-position j' on machine k and in order-position j" on machine k_2. For the schedule to be feasible

$$h_{j'}^{(k_1)} + t_i^{(k_1)}\, x_{ij}^{(k_1)} \leq h_{j''}^{(k_2)} \tag{4.2.7}$$

Given the specific order of production for item i, (4.2.7) guarantees that machining on (k_2) is not commenced until machining on (k_1) is completed. But we cannot add (4.2.7) in its present form as a constraint, since it requires too much, viz., that the starting time of the j"-positioned item on machine k_2 never precedes the finishing time of the j'-positioned item on

machine k_1; we want (4.2.7) to hold only whenever item i happens to be the item scheduled in both of these ordered positions. Therefore we merely need to require

$$h_{j'}^{(k_1)} + t_i^{(k_1)} x_{ij'}^{(k_1)} \leq h_{j''}^{(k_2)} + M(1 - x_{ij}^{(k_1)} + M(1 - x_{ij''}^{(k_2)}) \quad (4.2.8)$$

where M is a large positive integer and $h_j^{(k)}$ is evaluated by (4.2.6) we see that (4.2.8) is binding constraint only if

$$x_{ij'}^{(k_1)} = x_{ij''} = 1 .$$

In general, for item i, each pair of process stages p and p + 1, each ordered couple (machine k_1 in process stage p, machine k_2 in process stage p + 1) and j' = 1, 2, . . ., $n(k_1)$; j" = 1, 2, . . ., $n(k_2)$, we have a constraint of the form (4.2.8).

Secondly, we consider for item i the restriction that any machining taking place within process stage p must be on only one machine at a time. We suppose machines k_1 and k_2 appear in the i^{th} row and p^{th} column of the first matrix (or say matrix I).

Under the assumption $x_{ij'}^{(k_1)} = x_{ij''}^{(k_2)} = 1$, we require that one of the relations below must hold in order that the schedule be feasible:

$$h_{j'}^{(k_1)} + t_i^{(k_1)} x_{ij'}^{(k_1)} \leq h_{j''}^{(k_2)} \quad (4.2.9a)$$

$$h_{j''}^{(k_2)} + t_i^{(k_2)} x_{ij''}^{(k_2)} \leq h_{j'}^{(k_1)} \quad (4.2.9b)$$

Similar to the reasoning in (4.2.8), we want either (4.2.9a) or (4.2.9b) to hold only whenever item i happens to be the item scheduled in both of these ordered positions. Our linear constraint then is the pair

$$h_{j'}^{(k_1)} + t_i^{(k_1)} x_{ij'}^{(k_1)} \leq h_{j''}^{(k_2)} + M(1 - x_{ij'}^{(k_1)}) + M(1 - x_{ij''}^{(k_2)}) + \delta M \qquad (4.2.10a)$$

$$h_{j''}^{(k_2)} t_i^{(k_2)} x_{ij''}^{(k_2)} \leq h_{j'}^{(k_1)} + M(1 - x_{ij'}^{(k_1)}) + M(1 - x_{ij''}^{(k_2)}) - M(\delta - 1) \qquad (4.2.10b)$$

where δ is either 0 or 1.

In general for item i, each process stage p, each couple (machine k_1 in process stage p, machine k_2 in process stage p + 1) and $j' = 1, 2, \ldots, n(k_1)$, $j'' = 1, 2, \ldots, n(k_2)$, we have a constraint of the form (4.2.10).

Our objective function is the time at which processing on all the items has been completed. The optimal schedule will give earliest time. Let h* represent the earliest point in time according to a given schedule. We desire to find a schedule which minimizes h*. If, for example, all the items must be "finished off" on machine m, then the optimizing form simply is to minimize $\left(h_n^{(m)} + T\, x_n^{(m)} \right)$, where (4.2.6) is used to evaluate $h_n^{(m)}$.

This model is capable of handling a wide class of machine sequencing problems but the computation required is very large for this model to be of any practical use. There is hope for further developments.

4.3. Bowman's Formulation Giving Least Total Time

Another formulation is given by Bowman (12) which involves an even greater number of variables and constraints than Wagner's model, but no formal restrictions, including the size of the problem, are inherent in this method. The scheduling problem in its most simple form consists of a number of jobs to be done on a number of machines, each having a number of operations to be performed by the various machines in a specified sequence. What feasible schedule covers the least total time?

The Problem. Let the jobs be J_1, J_2, and J_3, and machines M_1, M_2, M_3, and M_4, and the time periods (small) run from 1, 2, 3, . . ., T_p.

The times required (in time period units) are:

	M_1	M_2	M_3	M_4
J_1	5	2	8	7
J_2	4	3	8	5
J_3	7	0	0	6

Fig. 4.4.

4.3.1. The Problem Constraints. The basic variables in the formulation are of the nature $J_1^{M_1:1}$ meaning product (job) J_1 having machine operation M_1 during time period 1. All these variables are to take the values zero or one in the solution; i.e., this process is or is not taking place during this time period. The form of constraints is:

$$1 \geqslant J_1^{M_1:1}, J_1^{M_1:2}, \ldots, J_1^{M_1:T_p}, J_1^{M_2:1}, \ldots, J_1^{M_2:2}, \ldots,$$

$$J_2^{M_1:1}, \ldots, J_2^{M_4:T_p}, \ldots, J_3^{M_4:T_p} \quad 0$$

It is necessary to include constraints assuring that the individual operations will be performed. For instance, J_1 requires five time units of processing on machine M_1. The form of constraints is:

$$\sum_{i=1}^{T_p} J_1^{M_1:i} = 5 \qquad \sum_{i=1}^{T_p} J_1^{M_2:i} = 2 \quad \ldots$$

$$\sum_{i=1}^{T_p} J_2^{M_1:i} = 4 \quad \ldots \qquad \sum_{i=1}^{T_p} J_3^{M_4:i} = 6$$

Two or more products may not be processed by the same machine at the same time, so conflicting assignments are forbidden. The form of constraints is:

$$J_1^{M_1:1} + J_2^{M_2:1} + J_3^{M_1:1} \leqslant 1$$

$$J_1^{M_1:2} + J_2^{M_1:2} + J_3^{M_1:2} \leqslant 1$$

$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$$

$$J_1^{M_4:T_p} + J_2^{M_4:T_p} + J_3^{M_4:T_p} \leqslant 1$$

The main part of the problem is proper sequencing. No operation may be undertaken until the previous operation on the product in the specified sequence has been completed in a previous time period. For example, product J_1 requires five time units on machine M_1 before its operation on M_2 can be

started. This operation on machine M_2 (two time units), in turn, must precede the operation on machine M_3. The form of constraints is:

$$5J_1^{M_2:z} \leq \sum_{i=1}^{z-1} J_1^{M_1:i} ,$$

$$2J_1^{M_3:z} \leq \sum_{i=1}^{z-1} J_1^{M_2:i} ,$$

. . . . . .

$$6J_3^{M_1:z} \leq \sum_{i=1}^{z-1} J_3^{M_4:i} ,$$

for all $z = 1$ to $z = T_p$.

4.3.2. The Objective Function. To find a solution to the L. P. problem, the variables, of the form $J_1^{M_1:1}$, must have values associated with them (many such values may be zero). In a sense, the objective is to have the final operations on all the products performed as early as possible. Prior operations such as those on machine M_3, will of course have preceded the final operations. The following objective function is suggested to be minimized.

$$\begin{aligned}\text{Objective function} = {} & 1(J_1^{M_4:23} + J_2^{M_2:23} + J_3^{M_1:23}) \\ & + 4(J_1^{M_4:24} + J_2^{M_2:24} + J_3^{M_1:24}) \\ & + 16(J_1^{M_4:25} + J_2^{M_2:25} + J_3^{M_1:25}) \\ & + 64(J_1^{M_4:26} + J_2^{M_2:26} + J_3^{M_1:26})\end{aligned}$$

$$\vdots \qquad \vdots \qquad \vdots$$

$$+ K_{T_p}(J_1^{M_4:T_p} + J_2^{M_2:T_p} + J_3^{M_1:T_p})$$

where $K_{T_p} = 4K_{T_{p-1}}$. The rationale of the objective function is that it makes operations (the last one on each product) toward the end of the time periods costly. The number of time periods chosen in advance of solution may certainly be equal to or less than the simple sum of all operation times (55), and can be no less than the sum of operation times required on the longest product (22). The cost associated with any operation in a time period is a synthetic one equal to the sum of all prior costs plus one. This exploding cost function thus forces operations toward the beginning for economic reasons.

This model is also impractical due to computation problem. This simple problem presented has 300 to 600 variables, depending on the number of time periods chosen the number equals (products) x (machine) x time periods . The number of constraints will be even greater than this. The author does not make any claim of the practicality of the method.

4.4. A Compact Formulation by Manne

Manne's (56) formulation of integer L. P. model is most compact of the three formulations and it might be computationally practical in some cases. A proposal for the integer L. P. to the typical job-shop scheduling problem is given. It involves both sequencing and noninterference restrictions. It is assumed that this sequencing problem involves performance

of 'n' tasks--each task is defined in such a way as to require the services of a single machine for an integral number of time units (say days). The problem is to draw up a plan for time phasing the individual jobs so as to satisfy (1) sequencing requirements, e.g., children must be washed before dried, and (2) equipment interference problems, e.g., the one-year old and three-year old cannot occupy the same bathtub at the same time. (All parents will hope that each of these tasks can be performed in less than a day.) The integer valued unknowns x_i are to indicate the day on which the task i is begun (x_u = 0, 1, 2, . . ., D). A schedule is to be drawn up so as to minimize the 'make span', i.e., the elapsed calendar time for the performance of all jobs--subject, of course, to the constraints upon sequencing and machine interference and also subject to any delivery date requirements on individual items.

4.4.1. Noninterference Restrictions. Let the jobs u and v require a_u and a_v consecutive days, respectively. Then if they are to be prevented from occupying the same machine at the same time, we must require that one of the two must precede the other by sufficient time so that the first one can be completed before the second is begun; either

$$x_u - x_v \geq a_v, \text{ or else } x_v - x_u \geq a_u \qquad (4.4.1)$$

To convert this condition into a linear inequality in integer unknowns we define a new integer valued variable y_{uv} and write down the following restrictions.

$$0 \leq y_{uv} \leq 1 \qquad (4.4.2)$$

$$(D + a_v)y_{uv} + (x_v - x_u) \geq a_u \qquad (4.4.3)$$

$$(D + a_u)(1 - y_{uv}) + (x_u - x_v) \geq a_v \qquad (4.4.4)$$

Condition (4.4.2) insures that $y_{uv} = 0$ or 1. We already know that $x_u - x_v \geq D$. The conditions (4.4.3) and (4.4.4) have the following effect. If

$$(x_u - x_v) \quad \left\{ \begin{array}{c} < \\ = 0 \\ > \end{array} \right\} \quad \left\{ \begin{array}{c} 0 \\ 0 \\ 0 \end{array} \right\}$$

then

$$y_{uv} = \left\{ \begin{array}{c} 0,\ 1 \\ 1 \\ 1 \end{array} \right\} \quad \text{and} \quad y_{uv} = \left\{ \begin{array}{c} 0 \\ 0 \\ 0,\ 1 \end{array} \right\}$$

where the first set of values for y_{uv} is implied by condition (4.4.3) and the second set by condition (4.4.4).

Hence it is seen that if $(x_u - x_v) = 0$, there is no value that can be assigned to y_{uv} so as to satisfy both (4.4.3) and (4.4.4). If $(x_u - x_v) \neq 0$, y_{uv} will be set at a value of either zero or unity, depending upon which job precedes the other.

4.4.2. Sequencing Restrictions. If the job u is to precede job v, this means that job v is to be performed at least a_u days later than u. The integer programming condition becomes

$$x_u + a_u \leq x_v \qquad (4.4.5a)$$

'Weak' precedence relations may be written in an analogous fashion. For instance, in order to specify that both jobs i

and u precede v, but that there is no precedence restriction affecting the performance of i and u, we would have

$$x_i + a_i \leq x_v \qquad x_u + a_u \leq x_v \tag{4.4.5b}$$

There might be a delay of exactly L_{jk} days between the performance of jobs u and v. For this restriction we will have

$$x_u + a_u + L_{uv} = x_v \tag{4.4.5c}$$

4.4.3. Specific Delivery Requirements. If the shop is committed to the delivery of an individual job no later than a specified date and if task u is the last task which the shop is to perform upon the item and if the item is to be available on day d_u, this form of requirement may be written

$$x_u + a_u \leq d_u \tag{4.4.6}$$

4.4.4. Overall Delivery Requirements. We shall employ our minimum as "make span" or total calendar time required for the completion of all the jobs. If this calendar time is denoted by t, the problem now consists of the minimization of t with respect to nonnegative integers x_u and y_{uv}, subject to constraints (4.4.2) to (4.4.6), and also subject to

$$x_u + a_u \leq t \qquad (u = 1, \ldots, n) \tag{4.4.7}$$

From the economic point of view, we should minimize the dollar cost. But in minimizing "make span" we receive the following cost and profit benefits: (1) A lowered amount of inventory tied up in work in process, (2) a shorter average customer delay time, and (3) a lower amount of idle time incurred prior to the performance of all currently booked jobs--i.e., a greater capacity to take on additional work as new orders materialize.

4.4.5. Computations. If all the slack variables and also the minimand t are excluded, the number of unknowns here is equal to the total number of x_u plus y_{uv}. If, then, there are tn tasks and also p_m possible conflicting pairs of machine assignments, the total number of unknowns would be $n + p_m$. For example, we have five machines with ten tasks to perform on each; here $n = 50$ (5 x 10) and $p_m = \frac{1}{2}(5)(10)(10 - 1) = 225$. The total number of integer valued unknowns x_u and y_{uv} would come, therefore, to 275, an impressive computational load but by no means an impossible one.

If Wagner's formulation is used, the total number of unknowns would come to about 600; again, slack variables and also make span minimand are neglected. In general, Wagner's formulation will require slightly more than twice the number of unknowns than in Manne's proposal.

5.0. THE USE OF SCHEDULE ALGEBRAS IN JOB-SHOP SCHEDULING

Most of the work in mathematical solution of scheduling problems has been done by Giffler (21, 23, 24). We will show more efficient ways to solve production scheduling problems and how to simplify and reduce the effort required to write computer programs. We will give some of the theory of scheduling as given by Giffler (21) and then show how the theory is used to define and solve the production scheduling problem which cannot be solved by most other techniques available at the present time.

5.1. Precedes and Next Precedes Relations

In every scheduling problem there is an order system whose elements are jobs to be scheduled. The basic order relation, which connects the jobs or tasks, is called a precedes relation and is designated by the symbol $\trianglelefteq$. The word precedes here has the same meaning as "must start before or at the same time". The statement $a \trianglelefteq b$ means task a precedes task b.

The precedes relation $\trianglelefteq$ includes the relation next precedes, designated by the symbol $\vartriangleleft$. The statement $a \vartriangleleft b$ is taken to mean that task "a" next precedes task "b", or, more specifically, that there exists a transitive chain of relations $\trianglelefteq$ from a to b, which includes no other task as an intermediary. Each precedes relation $\trianglelefteq$ contains one or more chains of zero or more next precedes relations $\vartriangleleft$. In Fig. 5.1, the relation $a \trianglelefteq d$ consists of two chains of relations $\vartriangleleft$, namely,

a $\lessdot$ b $\lessdot$ c $\lessdot$ d and a $\lessdot$ d. The first of these chains is said to be of level 3 and the second is said to be of level 1 since it has one relation $\lessdot$. Each precedes relation of a task to itself is said to consist of one chain of level zero.

Each chain of (zero or more) relations $\lessdot$ is quantified by associating with it a number which is the minimum interval of time necessary to traverse the chain. Thus with a one-level chain a $\lessdot$ b, we would associate a number which is the minimum time after task "a" has started before task "b" can start.

We express the fact that task i does not next precede task j by the notation

$$s_{ij}^{(1)} = 0$$

The superscript (1) tells us we are talking about one-level chains; the symbol 0 tells us that there is no one-level chain from i to j, or, in other words, that i does not next precede j. If task i does next precede j, we would write that

$$s_{ij}^{(1)} = t_{ij}$$

where t_{ij} is a minimum interval of time, after task i has started, before task j can start. When t_{ij} has zero magnitude we write $s_{ij}^{(1)} = \smile$, and expression $s_{ij}^{(1)} = 0$ means that i $\not\lessdot$ j. The number $\smile$ has immediate application in quantifying all chains of level 0. We simply say that

$$s_{ij}^{(0)} = \quad , \qquad \text{if } i = j$$
$$= 0 , \qquad \text{if } i \neq j$$

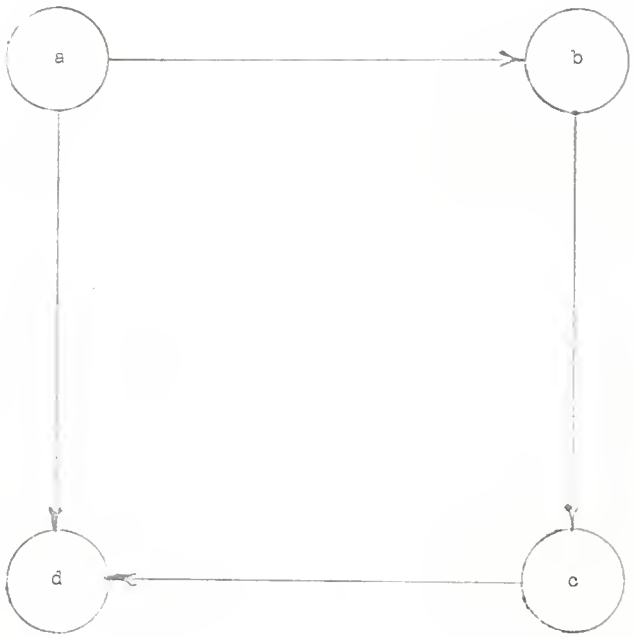

Fig. 5.1.

The above does not hold for chains of level greater than zero. We introduce the concepts of the "set of all chains of given level" which connect two tasks. Set of all chains of level 1 point to j is written as $(s_{ij}^{(1)})$. This set will contain the number zero if $i \not\ll j$, or it will contain one or more positive numbers t_{ij}'s if $i \ll j$. We express the above as

$$(s_{ij}^{(1)} = (t_{ij}\text{'s}) \qquad i = j$$
$$= (0) \qquad i \not\ll j$$

The set of the times to traverse all chains of level 2 can be obtained easily from the sets of the times to traverse all chains of level 1.

We write that

$$(s_{ij}^{(2)}) = (t_{ik} + t_{kj})$$

for all k such that $i \ll k \ll j$.

Another and potentially more useful way to summarize the calculation of the set $(s_{ij}^{(2)})$ is to write that

$$(s_{ij}^{(2)}) = \sum_{k} (s_{ik}^{(1)}) \odot (s_{kj}^{(1)})$$

where $\odot$ is a symbol for multiplication, and multiplying $(s_{ik}^{(1)})$ and $(s_{kj}^{(1)})$ means to add each entry in $(s_{ik}^{(1)})$ to all entries in $(s_{kj}^{(1)})$ when they are both nonzero, and the summation over k means to collect all the products into one set.

5.2. Schedule Algebras

We will discuss two special algebras for scheduling problems. The first of these two is equivalent to the conventional matrix algebra and is called "schedule algebra", and the second is equivalent in terms of its postulates to the conventional matrix algebra of nonnegative matrices. The second algebra is called "schedule* algebra".

The elements of schedule algebra are rectangular matrices whose $(i, j)^{th}$ entries are sets. We refer to the elements of the algebra as matrices and enclose all symbols for matrices in brackets. We refer to the sets as sets or element-sets and enclose symbols for these sets in braces. We reserve the term element to refer only to the individual entries in the sets. Symbols for elements are never enclosed. The set $\{0\}$, which contains only the number zero, is said to be empty. A nonempty set may contain any arithmetic numbers except that no two numbers may have precisely the same magnitude and opposite sign. A number of zero magnitude is to be replaced by ± 0.

For addition of matrices $[A]$ and $[B]$, we use $\oplus$ symbol and it is done in a manner similar to the conventional algebra. Precisely to add two sets we perform the following operations.

1. Collect the entries of both sets.
2. Replace by a zero all pairs of entries which have the same magnitude and opposite sign.
3. If an entry remains which is not zero, suppress all zeros. If all remaining are zeros, suppress all zeros but one.

Example.

$$\begin{bmatrix} \{1,\ -\iota\} & \{-3\} \\ \{0\} & \{0\} \end{bmatrix} + \begin{bmatrix} \{-1\} & \{3\} \\ \{\iota\} & \{-\iota\} \end{bmatrix}$$

$$= \begin{bmatrix} \{1,\ -1,\ -\iota\} & \{-3,\ 3\} \\ \{0, \iota\} & \{0\ \ -\iota\} \end{bmatrix} = \begin{bmatrix} \{-\iota\} & \{0\} \\ \{\iota\} & \{-\iota\} \end{bmatrix}$$

Matrix multiplication in schedule algebra resembles conventional matrix multiplication and is designated by the symbol $\odot$. When there is no ambiguity we may write $[A] \odot [B]$ as $[A][B]$; note here that $[A]$ has as many columns as $[B]$ has rows. When multiplying elements the following rule is observed.

$$\begin{aligned} a \odot b &= |a| + |b| = c && \text{if a and b have the same sign} \\ &= -c && \text{if a and b have the opposite sign} \\ &= 0 && \text{if a or b is zero} \end{aligned}$$

The plus sign in the above rule indicates a conventional addition operation. Thus $\iota \odot a = a$ for all a and $-\iota \odot a = -a$ for all $a \neq 0$.

Example. ($[A]$ has two columns and $[B]$ has two rows).

$$\begin{bmatrix} \{\iota,\ -1\} & \{1,\ 1\} \end{bmatrix} \odot \begin{bmatrix} \{\iota,\ 1\} & \{0\} \\ \{\iota\} & \{-1\} \end{bmatrix}$$

$$= \begin{bmatrix} \{\iota,\ 1,\ -1,\ -2,\ 1,\ 1\} & \{0,\ 0,\ -2,\ -2\} \end{bmatrix}$$

$$= \begin{bmatrix} \{\iota,\ 0,\ -2,\ 1,\ 1\} & \{-2,\ -2\} \end{bmatrix}$$

$$= \begin{bmatrix} \{\iota,\ -2,\ 1,\ 1\} & \{-2,\ -2\} \end{bmatrix}$$ (suppressing the zero)

Matrices in schedule algebra obey, with respect to $\oplus$ and $\odot$, the same postulates as do matrices in conventional matrix algebra. The identity matrix in addition is written $[0]$; all of its sets are $\{0\}$; that is, empty.

The identity matrix in multiplication is the square matrix $[I]$ having sets $\{\iota\}$ on the main diagonal and $\{0\}$ elsewhere. The matrix $[-I]$ is defined the same as $[I]$ except that its diagonal sets are $\{-\iota\}$. Premultiplying and postmultiplying any matrix by $[-I]$ serves to change the sign of all nonzero elements in the matrix. Thus schedule algebra subtraction is defined as follows.

$$[A] \ominus [B] = [A] \oplus [-I] \; [B]$$

All the theorems in conventional algebra which depend solely on the postulates of the algebra apply also in schedule algebra. In certain problems it is convenient to change the rule for matrix addition so that it becomes a maximizing operation. Then our schedule algebra becomes schedule* algebra. Matrices in schedule* algebra have entries which are either zero or a nonnegative number (including iota). The symbol for schedule* algebraic addition is *. The $(i, j)^{th}$ entry of the sum $[A] * [B]$ is $\max(a_{ij}, b_{ij})$, where a_{ij} and b_{ij} are the $(i, j)^{th}$ entries of $[A]$ and $[B]$ respectively.

The number 0 in these maximizations is treated as though it were negative infinity. The identity matrix for addition in schedule* algebra is written $[0]$ and has zero entries.

Example.

$$\begin{bmatrix} 1 & 0 \\ \iota & 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} \max(1, 0) & \max(0, 1) \\ \max(\iota, 1) & \max(1, 5) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 5 \end{bmatrix}$$

In this case multiplication of matrices is exactly the same as in schedule algebra except that all additions which occur in the matrix multiplications are to be carried out according to the maximization rule, defined above. The identity matrix in multiplication is the square matrix $[I]$ having entries ι on its main diagonal and 0 elsewhere. Schedule* algebraic multiplication of matrices is denoted by the symbol $\#$.

Example.

$$\begin{bmatrix} 1 & 0 \\ \iota & 1 \end{bmatrix} \# \begin{bmatrix} 0 & 1 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} \max(1 \odot 0, 0 \,.\, 1) & \max(1 \odot 1, 0 \,.\, 3) \\ \max(\iota \odot 0, 1 \,.\, 1) & \max(\iota \odot 1, 1 \,.\, 3) \end{bmatrix}$$

$$= \begin{bmatrix} \max(0, 0) & \max(2, 0) \\ \max(0, 2) & \max(1, 4) \end{bmatrix} = \begin{bmatrix} 0 & 2 \\ 2 & 4 \end{bmatrix}$$

5.3. Schedule Algebraic Formalizations

We define one-level (or next precedes) matrix S , whose $(i, j)^{th}$ set

$$\begin{aligned} s_{ij}^{(1)} &= (t_{ij}\text{'s}), && \text{if } i \lll j \\ &= (0) && \text{if } i \not\lll j \end{aligned} \qquad (5.3.1)$$

and zero level precedes matrix S^0 be square matrix with $(i, j)^{th}$ set

$$\left\{ s_{ij}^{(0)} \right\} = (\mathcal{L}), \quad \text{if } i = j$$
$$= (0), \quad \text{if } i \neq j \qquad (5.3.2)$$

$\left[S^0 \right]$ is, of course, the identity matrix $\left[I \right]$.

The set $\left\{ s_{ij}^{(2)} \right\}$ of all two-level chains from task i to task j is the schedule algebraic sum of all k^{th} products, $\left\{ s_{ik}^{(1)} \right\} \odot \left\{ s_{kj}^{(1)} \right\}$; i.e.,

$$\left\{ s_{ij}^{(2)} \right\} = \sum_{k} \left\{ s_{ik}^{(1)} \right\} \odot \left\{ s_{kj}^{(1)} \right\} \qquad (5.3.3)$$

The above concept can be extended to general w level chains from i to j. We may write

$$\left\{ s_{ij}^{(w)} \right\} = \sum_{k} \left\{ s_{ik}^{(w-k)} \right\} \odot \left\{ s_{kj}^{(k)} \right\} \qquad (5.3.4)$$

where k is nonnegative integer such that $k \leq w$. We designate the set of lengths of all chains (of any level) from i to j by symbol θ_{ij} , and write that

$$\left\{ \theta_{ij} \right\} = \left\{ s_{ij}^{(0)} \right\} \oplus \left\{ s_{ij}^{(1)} \right\} \oplus \ldots \oplus \left\{ s_{ij}^{(\lambda *)} \right\} \qquad (5.3.5)$$

where $\lambda *$ is the maximum number of relations $\lessdot$ in any chain from i to j. We assemble the $\left\{ \theta_{ij} \right\}$ into the matrix $\left[\theta \right]$ and write that

$$\left[\theta \right] = \left[I \right] \oplus \left[S^1 \right] \oplus \left[S^2 \right] \oplus \ldots \oplus \left[S^{\lambda} \right] \qquad (5.3.6)$$

where λ is maximum of all $\lambda *$. It is clear that $\left[S^k \right] = \left[0 \right]$ for all $k > \lambda$. Also it is shown that equation (5.3.6) may be written as

$$\left[\theta\right] = \left[I \ominus S\right]^{-1} \tag{5.3.7}$$

The matrix $\left[\theta\right]$ summarizes all the restrictions on the starts of j^{th} tasks which are the consequences of the relations $\ll$ and the time intervals associated with these relations. Each set $\{\theta_{ij}\}$ contains (if not empty) the lengths of time to traverse all chains of relations $\ll$ from i to j, and is a lower bound on the closeness of the starts of i to j. The maximum entry in the set is denoted by the symbol $\emptyset_{ij}$. It is the greatest of the lower bounds and represents the shortest possible interval of time which can separate the tasks' respective starts.

For the cases in which one needs to know only the maximum chains connecting $(i, j)^{th}$ pairs of tasks, it is possible to define a schedule* algebraic matrix $\left[\emptyset\right]$, whose $(i, j)^{th}$ entry is the number $\emptyset_{ij}$ (defined already), and to solve for this matrix by using the equation

$$\left[\emptyset\right] = \left[I\right] * \left[S^1\right] * \left[S^2\right] * \ldots * \left[S^{\lambda}\right]$$

or, since $S^{\lambda+1} = \left[0\right]$

$$\left[\emptyset\right] = \left[I\right] * \left[S\right] \# \left[\emptyset\right] \tag{5.3.8}$$

In the above equations, an entry in $\left[S\right]$ is assumed to contain, at most, one element, zero, if the set is empty, or the maximum of the set. This assumption is implied in the use of schedule* algebraic formalizations and is the reason that each set is represented always by the entry it contains.

5.4. Determinate Scheduling Problems

The two basic types of scheduling problems considered are (1) determinate problems, and (2) indeterminate problems.

The determinate problems are those problems in which the desired answers are implicit in the given data, e.g., problems to determine start times of tasks if the order relations which connect them and the time required to perform each task are given. Indeterminate problems have insufficient data to determine the answers and for which it is necessary to define functions to generate the missing data, e.g., production scheduling problems in which the order of performing tasks on facilities is not given a priori but must be determined by the repeated application of a schedule rule.

The determinate problem, as a type of problem, is similar to a problem of solving a set of simultaneous equations in that the answer (namely, the values of the unknowns) is implicit in the equations.

5.4.1. Example of a Determinate Scheduling Problem. We are given a set of n tasks and are told that certain pairs of these tasks must be performed consecutively (i.e., without a third task intervening), and we are given for each of these pairs the minimum time interval, after the first task starts, before the second can start. We are told, finally, when each unpreceded task starts and are asked to find when all the tasks start.

We note first that we are given the matrix $\lceil S \rceil$ and a row vector $\lceil T_0 \rceil$, whose j^{th} entry, $t_j^{(0)}$, is the start time of task j,

if j is unpreceded, and is zero otherwise. If we let $[T]$ with j^{th} entry t_j be a row of the starting times which are to be determined, then we can write

$$[T] = [T_0] \# [\emptyset]$$
$$= [T_0] \# [I * S^1 * S^2 * \ldots * S^\lambda]$$
$$= [T_0] * [T_0 S] * [T_0 S][S] * [T_0 S^2][S] * \ldots * [T_0 S^{\lambda-1}][S] \tag{5.4.1}$$

By making use of a special property of schedule* algebraic addition, namely, that

$$[A] * [A] = [A] \text{ for all } [A]$$

the equation (5.4.1) can be put into the following form

$$[T] = [T_\lambda]$$

where

$$[T_\lambda] = [T_{\lambda-1}] * [T_{\lambda-1}][S] = [T_{\lambda-1}][I * S]$$

and

$$[T_1] = [T_0] * [T_0][S] = [T_0][I * S] \tag{5.4.2}$$

To solve the given problem, we need only to solve equation (5.4.2) and for this we need a computing algorithm. The following algorithm is recommended by Giffler (61).

Step 1. Prepare a sequence of n boxes (or words) to be called T. Place the given start time of an i^{th} task in the i^{th} box of T. Place a zero in all other boxes of T. (The algorithm, as it precedes, will eventually place the start of each task in its corresponding box

in T.)

Step 2. Prepare a triplet of boxes for each given the next-precedes relation. In the first box of each triplet place the index i of the task which next-precedes; in the second box place the index j of the task which is next preceded; in the third box place the given minimum time t_{ij}, after the first task starts, before the second can start. Sequence the triplets, j indices within i indices. Call this set of boxes N.

Step 3. Set an index K = 0 and place this index in (or next to) each i^{th} box in T which contains a nonzero entry. Call the number attached to an i^{th} box the K_i number of the box.

Step 4. Note the "left most" box in T with $K_i = K$; say it is the i^{th} box. (Initially, all nonzero boxes will have $K_i = 0$).

Step 5. Add the entry in the i^{th} box above to the first t_{ij} in N. Compare the sum with the entry in the j^{th} box of T. If the sum is greater, replace the entry in the j^{th} box by the sum and check the box. Repeat this step with each j^{th} successive t_{ij}.

Step 6. Repeat from step 3 until there are no boxes with $K_j = K$. When this happens, increase the K_i number of all checked boxes by one and uncheck; advance K to K + 1 and repeat from step 4. If there are no checked boxes, transfer to OUT.

OUT. When this step is reached, the i^{th} box in T will contain the start time of the i^{th} task.

5.5. Indeterminate Scheduling Problems

We say that a scheduling problem is determinate if the answer which is sought is implicit in the given restraints of the problem. A problem in contrast is indeterminate if the restraints do not imply the answer, but at most a set of possible (or feasible) answers. To solve an indeterminate problem one must first make it determinate. This is customarily accomplished in scheduling problems by specifying a rule for selecting next tasks to be performed; that is, when the next task is not prescribed by (or implicit in) the given constraints. We consider the indeterminate scheduling problems in the following context.

1. There are n tasks to be performed. For each task there is specified a group of facility types which is needed to perform the task. There may be one or more facilities of each type available to perform the task.
2. Certain pairs of tasks must be performed consecutively. For each of these pairs there is a given minimum time interval after the first task starts before the second can start.
3. Certain pairs of tasks, because they are to be performed by the same facility type and possibly by the same facility of the given type (i.e., on the same machine), may be performed consecutively. For each of these pairs there is a given number which is the minimum

delay, after the first task in the pair starts, before the second can start, if we assume, of course, that the second task ultimately next follows the first task on the same facility.

4. For some tasks there will be given an earliest time at which the tasks may be started.
5. For each task the time (a number) for completion may be given.

The problem is to determine for each task, the particular facilities which perform it and the time it starts. Also, if time to perform the task to completion is given, the ending time is determinable. To solve the problem we assume a rule is given for determining which task is to be performed next. We will give a mathematical solution to the problem using schedule* algebra. We start by collecting information in items 1, 2, and 3 (above) into an open S_0 matrix whose $(i, j)^{th}$ entry

$$\begin{aligned} s_{ij} &= t_{ij} && \text{if it is given that } i \lessdot j \\ &= t_{ij},\ x_{ij} && \text{if it is possible that } i \lessdot j \\ &= 0 && \text{if it is impossible that } i \lessdot j \end{aligned} \qquad (5.5.1)$$

The variable x_{ij} is given the value 1 if tasks i and j are to be performed consecutively by a same facility; otherwise, x_{ij} is set equal to 0. S_{ij} is set equal to 0 at the outset, if it is given that $i = j$, that $j \leqslant i$ or $j \lessdot i$, or if it is given that i and j are to be performed by facilities of different types, and it is not given that $i \lessdot j$. The information in item 4 (above) is assembled into a row vector $[T_0]$ whose j^{th}

entry is the earliest permitted start time of j, if this is specified, and is zero otherwise.

The main use of information in item 5 (above) is to determine the completion time of tasks if their starting times are given. This information does not require a special treatment.

Let us make a few assumptions for simplification.

1. There is exactly one facility of each type.
2. Each task requires exactly one of these facilities for its performance.
3. The next tasks are selected to be performed by these facilities by the FOFO (first off, first on) rule. In our problem this rule means that each task selected to be performed next by a facility must be that particular task which will first make the facility (which performs it) available to perform another task.

An iterative procedure for determining the starting time of all tasks is given below. This procedure is imitative of the one used in the case of determinate scheduling problem. See equation (10).

Step 1. Construct the vector $[T_0]$ and the matrix as described above.

Step 2. Locate null or potentially null columns in $[S_0]$. A column is potentially null if it could be made null by setting all x's (defined previously) in the column equal to zero and striking all entries in rows of tasks previously selected to start.

Step 3. Determine for each j^{th} task whose column was

located in step 2, a "test number" equal to the j^{th} entry of the vector $\lceil T_0 \rfloor$, increased by minimum coefficient of an x in the j^{th} tasks row in $\lceil S_0 \rfloor$. These computed numbers, called FACATS (facility available time), are the earliest times that the facility which performs j could be available to start a next task, if j were to be selected to start next. Select j^{th} task whose calculated FACAT is minimum.

Step 4. Updata $\lceil S_0 \rfloor$ to $\lceil S_1 \rfloor$ as follows: (a) Set each i^{th} x_{ij} in column j equal to zero; (b) set each x_{jk} (if there are any) equal to y_{jk} (this change of variable is explained in the following paragraph); (c) if there is y_{hj}, set it equal to and all other y_{hk} (if any) equal to zero.

Step 5. Compute $\lceil T_1 \rfloor = \lceil T_0 \rfloor * \lceil T_0 \# S_1 \rfloor$. In multiplying $\lceil S_1 \rfloor$ by $\lceil T_0 \rfloor$, treat all x_{ij} as though they were 0, all y_{ij}'s as though they were iota.

The variables y are of transistory nature. They represent the value which x_{ij} may assume on its way to becoming an iota or a zero. When j^{th} task is selected to start, we know immediately the task which can no longer next precede it. We do not, however, know the particular task (if any) which will next follow it. This is the reason for the introduction of y's since we only know that (possibly) one of the x_{jk} will be set equal to iota and the others to zero. While the x_{jk}'s are in this "status", we give them the name y_{jk}'s.

5.5.1. A Numerical Example. The following numerical example will make the procedure clearer.

Two products are to be manufactured. The first product is assumed to have been released at time iota; and second at time 3. Each product requires for its manufacture the performance of three tasks. The interrelation of the tasks and their relevant processing times are depicted in the flow diagrams of Figs. 5.2 and 5.3.

In the figures the numbers in the nodes identify the tasks. The ordered triplet of numbers, which is placed to the right of each node, has the following meaning. The first number gives the facility-type needed to perform the task, the second number is the minimum time, after task i starts, before the next task can start, and the third number gives the minimum time, after i starts, before the next task can follow i on the same facility.

Let us first construct the vector $[T_0]$ and the matrix $[S_0]$ such that i^{th} row and/or column represents i^{th} task. Each x in $[S_0]$ is a different variable. (See Figs. 5.2 and 5.3.)

$$[T_0] = [\iota \quad \upsilon \quad 0 \quad 3 \quad 0 \quad 0]$$

$$[S_0] = \begin{bmatrix} 0 & 0 & \max(2,1x) & 0 & 1x & 0 \\ 0 & 0 & 1 & 1x & 0 & 1x \\ 2x & 0 & 0 & 0 & 2x & 0 \\ 0 & 1x & 0 & 0 & 1 & 1x \\ 1x & 0 & 1x & 0 & 0 & 2 \\ 0 & 2x & 0 & 2x & 0 & 0 \end{bmatrix} \qquad (5.5.2)$$

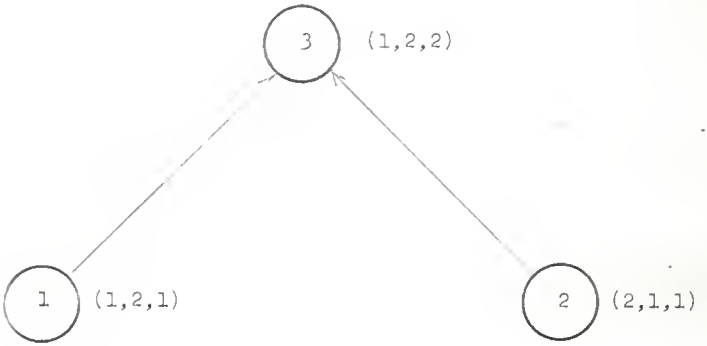

Fig. 5.2.

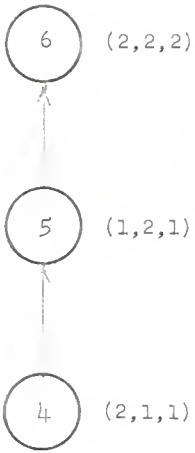

Fig. 5.3.

Having constructed $[S_0]$, we note that tasks 1, 2, and 4 are potentially null since all elements have an x factor, and have respective FACATS of 1, 1, and 4. Since tasks 1 and 2 are tied, we select task 1 (arbitrary) to start. This act of section changes $[S_0]$ to

$$[S_1] = \begin{bmatrix} 0 & 0 & \max(2,1y) & 0 & 1y & 0 \\ 0 & 0 & 1 & 1x & 0 & 1x \\ 0 & 0 & 0 & 0 & 2x & 0 \\ 0 & 1x & 0 & 0 & 1 & 1x \\ 0 & 0 & 1x & 0 & 0 & 2 \\ 0 & 2x & 0 & 2x & 0 & 0 \end{bmatrix} \qquad (5.5.2)$$

All of the entries in the first column are zero; this was obtained by changing the entries with variable x's to zero. And for the first row, change the variable from x to y; the rest of the elements are as before.

We compute $[T_1] = [T_0] * [T_0 \# S_1]$ and obtain

$$[T_1] = [\,\cup \quad \cup \quad 2 \quad 3 \quad 4 \quad 0\,] \qquad (5.5.3)$$

We now note that the tasks with potential null columns are 2 and 4 (task 1 having started we overlook its row and column), and that their FACATS happen to remain 1 and 4, respectively. Consequently we select task 2 to start next. This leads us to $[S_2]$.

$$[S_2] = \begin{bmatrix} 0 & 0 & \text{max}(2,1y) & 0 & 1y & 0 \\ 0 & 0 & 1 & 1y & 0 & 1y \\ 0 & 0 & 0 & 0 & 2x & 0 \\ 0 & 0 & 0 & 0 & 1 & 1x \\ 0 & 0 & 1x & 0 & 0 & 2 \\ 0 & 0 & 0 & 2x & 0 & 0 \end{bmatrix} \qquad (5.5.4)$$

Computing $[T_2] = [T_1] * [T_1 \# S_2]$, we obtain

$$[T_2] = [1 \quad 1 \quad 2 \quad 3 \quad 4 \quad 6] \qquad (5.5.5)$$

The tasks with potential null columns are now 3 and 4 and have FACATS 4 and 4, respectively. We select task 3 to start and change $[S_2]$ to

$$[S_3] = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1y & 0 & 1y \\ 0 & 0 & 0 & 0 & 2y & 0 \\ 0 & 0 & 0 & 0 & 0 & 1x \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 2x & 0 & 0 \end{bmatrix} \qquad (5.5.6)$$

Computing $[T_3] = [T_2] * [T_2 \# S_4]$, we obtain

$$[T_3] = [1 \quad 1 \quad 2 \quad 3 \quad 4 \quad 6] \qquad (5.5.7)$$

This time the only task with a potential null column is task 4. Since there is only one selection, we will not compute FACATS. Thus task 4 is selected to start next and we change $[S_3]$ to

$$[S_4] = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2y & 0 \\ 0 & 0 & 0 & 0 & 1 & 1y \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad (5.5.8)$$

Finding $[T_4] = [T_3] * [T_3 \# S_4]$, we obtain

$$[T_4] = [\ \cup\ \ \cup\ \ 2\ \ 3\ \ 4\ \ 6\ ] \qquad (5.5.9)$$

The next task to start is task 5 and we get

$$[S_5] = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1y \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad (5.5.10)$$

which gives $[T_5] = [T_4] * [T_4 \# S_5]$, or

$$[T_5] = [\ \cup\ \ \cup\ \ 2\ \ 3\ \ 4\ \ 6\ ] \qquad (5.5.11)$$

The last task to be selected is 6. Its selection changes $[S_5]$ to

$$[S_6] = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad (5.5.12)$$

We compute $[T_6] = [T_5] * [T_5 \# S_6]$, or

$$[T_6] = [0 \quad 0 \quad 2 \quad 3 \quad 4 \quad 6]$$

$[T_6]$ is the start time vector $[T]$. Each j^{th} entry of $[T]$ is the starting time of the j^{th} task.

The solution of the above problem is presented as a Gantt chart in Fig. 5.4.

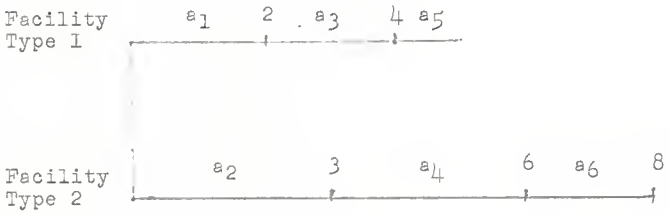

Fig. 5.4. a_i and b_i are, respectively, the times for which the facility types 1 and 2 are occupied by the job i.

6.0. SUMMARY

The job-shop scheduling problem is one of the most challenging problems which has been posed in operations research. So far algorithms have been developed for the simpler cases. Accurate solutions are available for problems not involving more than two or three machines. Of course, there is no limit to the number of jobs. We have reviewed the general problem of processing n jobs, not all identical, through m machines of different types. The processing time of each job on each machine and the order of scheduling each job through the machines are known. Different procedures used to determine the order in which the jobs should be fed through each machine, are developed under extremely simplifying assumptions. Moreover, the formulation of the problem itself is incomplete. We need procedures for which the objective function involving cost or profit is optimized instead of some function of time.

The total number of possible schedules for the above mentioned problem is $(n!)^m$, some of which, however, may not be feasible since the required operations must be performed in a specified order. The algorithms developed direct us to an optimal or approximately optimal schedule without enumerating all or most of the possible schedules.

In most of the formulations the criterion used for optimization is idle time or make span, i.e., the total time required to complete the processing of all the given jobs on the given set of facilities. This optimization means greater capacity

of the machines to take additional work. But this does not necessarily minimize the cost of production or maximize the profits. Thus the measure of effectiveness should be cost or profit.

Further research is required in the following areas:

1. Regressing back to a machine (in real situations, sometimes it is required to take the job back to a machine that has been previously used on that job). None of the procedures developed so far allows this.
2. Situations involving two machines such that the first machine could do what the second could do but the second could not do what the first could.
3. Sometimes the same job can be done on different machines but the processing cost on different machines may be different. This is a typical L. P. allocation problem without sequencing restrictions.
4. The job-shop scheduling problem should be treated as one involving three major variables; namely, jobs, machines, and operators. Where an operator may be trained to work on some or all of the machines.
5. In case of overtime work, which machines should be scheduled for overtime.
6. Balancing of overtime costs and delay penalties.
7. Another possible criterion may be to offer regular work to employees and minimize the overtime work. These days it is common to have some kind of contract in which no work pay is guaranteed. We should consider

this aspect at the same time.

8. More work should be done in the overlapping of production. In some situations it is possible to start production of a given job on the second machine before the completion of processing on the first machine.

So far the most fruitful approach to the complex scheduling problems which occur in reality seems to be that of simulation technique. It allows the experimentation of a system on paper. With the absence of a model describing the behavior of the system, we are not quite sure what outcome to expect if we change its operating conditions. We cannot take the risk of experimenting with the system itself. Thus simulation technique is used to test the decision rules without applying the decision rules to the real situations. The best set of decision rules is then picked up. The fruitfulness of this approach is enhanced by the availability of high-speed computing equipment.

ACKNOWLEDGMENT

The author wishes to extend sincere appreciation to his Major Professor, Dr. L. E. Grosh, for assistance given in the preparation of this report, and to Dr. F. A. Tillman, who suggested the topic and gave help and encouragement during its preparation.

The author's thanks are also extended to Mrs. Lola M. Crawford, who typed the report.

BIBLIOGRAPHY

1. Ackoff, R. L., B. L. Arnoff, and C. W. Churchman. Progress in Operations Research. New York: John Wiley and Sons, Inc., 1961.

2. Akers, S. B., and J. Friedman. "A Non-numerical Approach to Production Scheduling Problems," Journal of Operations Research Society of America, Vol. 3, November, 1955, pp. 429-442.

3. Akers, S. B. "A Graphical Approach to Production Scheduling Problems," Journal of Operations Research Society, Vol. 4, March, 1956, pp. 244-245.

4. Baker, C. T., and B. P. Dzielinski. "Simulation of a Simplified Job Shop," Journal of the Institute of Management Sciences, Vol. 6, January, 1960, pp. 311-323.

5. Banerjee, B. P. "Single Facility Sequencing with Random Execution Times," Journal of Operations Research Society, Vol. 13, May, 1965, pp. 358-364.

6. Barnes, W. E. "The Application of Computer Simulation to Production Scheduling Research," 16th National Meeting of the Operations Research Society of America.

7. Bellman, R. "Some Mathematical Aspects of Scheduling Theory," Journal of the Society of Industrial and Applied Mathematics, Vol. 4, September, 1956, pp. 168-205.

8. Bellman, R. E. "Combinatorial Processes and Dynamic Programming," Rand P-1284, February, 1958.

9. Bellman, R. E., and S. E. Dreyfus. "Applied Dynamic Programming," Princeton University, Princeton, New Jersey, 1962.

10. Bellman, R., and O. Gross. "Some Combinatorial Problems Arising in the Theory of Multi Stage Process," Journal of the Society of Industrial and Applied Mathematics, Vol. 2, September, 1954, pp. 175-183.

11. Blake, K. R., and W. S. Stopakis. "Some Theoretical Results on the Job Shop Scheduling Problem," Report M-1533-1, United Aircraft Corporation Research Department, East Hartford, Connecticut, July 1, 1959.

12. Bowman, E. M. "The Scheduling Sequencing Problem," Journal of Operations Research Society, Vol. 7, September, 1959, pp. 621-624.

13. Brooks, G. H., and C. R. White. "An Algorithm for Finding Optimal or Near Optimal Solutions to the Production Scheduling Problem," Journal of Industrial Engineering, Vol. 16, January, 1965, pp. 34-40.

14. Churchman, C. W., R. L. Ackoff, and E. L. Arnoff. "Introduction to Operations Research." New York: John Wiley and Sons, Inc., 1961.

15. Conway, R. W. "An Experimental Investigation of Priority Dispatching," Journal of Industrial Engineering, Vol. 11, June, 1960, pp. 221-230.

16. Dantzig, G. B. "A Machine Shop Scheduling Model," Journal of the Institute of Management Sciences, Vol. 6, January, 1960, pp. 191-196.

17. Dantzig, G. B. "On the Shortest Route through a Network," Journal of the Institute of Management Sciences, Vol. 6, January, 1960, pp. 187-190.

18. Dudek, R. A., and Teuton, Jr. "Development of M-Stage Decision Rule for Scheduling n Jobs through m Machines," Journal of Operations Research Society, Vol. 12, May, 1964, pp. 471-497.

19. Fisher, H., and G. L. Thompson. "Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules," Chapter 15 in reference (61).

20. Giffler, B. "Mathematical Solution of Explosion and Scheduling Problems," IBM Research Report RC-118, June 18, 1959. IBM Research Center, Business Systems Research, Yorktown Heights, New York.

21. Giffler, B. "Mathematical Solution of Production Planning and Scheduling Problems," IBM ASDD Technical Report 09.026, White Plains, New York, October, 1960.

22. Giffler, B. SIMPRO 1: An IBM 704-7090 "Simulation Program for Planning Scheduling and Monitoring Production Systems," IBM ASDD Technical Report, December, 1961.

23. Giffler, B. "Schedule Algebras and their Use in Formulating General Systems Simulation," Chapter 4 in reference (61).

24. Giffler, B., and G. L. Thompson. "Algorithms for Solving Production Scheduling Problems," IBM Research Report RC-118, Yorktown Heights, New York, June, 1959.

25. Giffler, B., and G. L. Thompson. "Algorithms for Solving Production Scheduling Problems," Journal of Operations Research Society, Vol. 8, July, 1960, pp. 487-503.

26. Giffler, B., G. L. Thompson, and V. Van Ness. "Numerical Experience with the Linear and Monte Carlo Algorithms for Solving Production Scheduling Problems," Chapter 3 in reference (61).

27. Giglio, R. J., and H. M. Wagner. "Approximate Solutions to the Three-machine Scheduling Problem," Journal of Operations Research Society, Vol. 12, March, 1964, pp. 305-324.

28. Gomory, R. E. "An Algorithm for Integer Solutions to Linear Programs," Princeton-IBM Mathematics Research Project, Technical Report No. 1, November 17, 1958.

29. Gomory, R. E. "All-integer Integer Programming Algorithm," Chapter 13 in reference (61).

30. Gomory, R. E. "Mixed Integer Programming Algorithm," unpublished Rand memorandum.

31. Hardgrave, W. W., and G. L. Nemhauser. "A Geometric Model and a Graphical Algorithm for a Sequencing Problem," Journal of Operations Research Society, Vol. 11, November, 1963, pp. 889-900.

32. Held, M., and R. M. Karp. "A Dynamic Programming Approach to Sequencing Problems," Journal of the Society of Industrial and Applied Mathematics, Vol. 10, March, 1962, pp. 196-210.

33. Held, M., R. M. Karp, and R. Shareshian. "Scheduling with Arbitrary Profit Functions," M. & A. -11, Data Systems Division, Mathematics and Applications Department, IBM Corporation, New York.

34. Heller, J. "Combinatorial, Probabilistic and Statistical Aspects of an M x J Scheduling Problem," Report NYO-2540, ABC Computed and Applied Mathematics Center, Institute of Mathematical Science, New York University, New York, February 1, 1959.

35. Heller, J. "Combinatorial Properties of Machine Shop Scheduling," NYO-2879, Atomic Energy Commission Research and Development Report, July, 1959.

36. Heller, J. "Some Numerical Experiments for an (M x J) Flow Shop and Its Decision Theoretical Aspects," Journal of Operations Research Society, Vol. 8, March, 1960, pp. 178-184.

37. Heller, J. "Some Problems in Linear Graph Theory That Arise in the Analysis of the Sequencing of Jobs through Machines," Report NYO-9847, AEC Computed and Applied Mathematics Center, Institute of Mathematical Science, New York University, New York, October, 1960.

38. Heller, J., and G. Logemann. "An Algorithm for the Construction and Evaluation of Possible Schedules," Journal of the Institute of Management Sciences, Vol. S, January, 1962, pp. 168-183.

39. IBM Group. "The Job Simulator: An IBM 704 Program," IBM Mathematics and Applications Department, International Business Machines Corporation, New York, New York, 1960.

40. Ignall, E., and L. Schrage. "Application of the Branch and Bound Technique to Some Flow Shop Scheduling Problems," Journal of Operations Research Society, Vol. 13, May, 1965, pp. 400-412.

41. Jackson, J. R. "Notes on Some Scheduling Problems," Management Sciences Research Project, Research Report No. 35, University of California, Los Angeles, October, 1954.

42. Jackson, J. R. "Scheduling a Production Line to Minimize Maximum Tardiness," Management Sciences Research Project, Research Report No. 43, University of California, Los Angeles, 1955.

43. Jackson, J. R. "An Extension of Johnson's Results on Job Lot Scheduling," Naval Research Logistics Quarterly Vol. 3, September, 1956, pp. 201-203.

44. Jackson, J. R. "Simulation Research on Job Shop Production," Naval Research Logistics Quarterly, Vol. 4, December, 1957.

45. Jackson, J. R. "Machine Shop Simulation Using SWAC: A Progress Report," Management Sciences Research Project, Discussion Paper No. 67, University of California, Los Angeles, April, 1958.

46. Jackson, J. R., and Y. Kuratani. "Production Scheduling Research: A Monte Carlo Approach," Management Sciences Research Project, Research Paper No. 61, University of California, Los Angeles. May, 1957.

47. Johnson, S. M. "Optimal Two- and Three-stage Production Schedules with Set-up Times Included," Naval Research Logistics Quarterly, Vol. 1, March, 1954, pp. 61-68, and Chapter 2 in reference (61).

48. Johnson, S. M. "Discussion," Journal of the Institute of Management Sciences, Vol. 6, April, 1959, pp. 299-303.

49. Karush, W. "A Counter-example to a Proposed Algorithm for Optimal Sequencing of Jobs," Journal of Operations Research Society, Vol. 13, March, 1965, pp. 323-325.

50. Karush, W., and L. A. Moody. "Determination of Feasible Shipping Schedules for a Job Shop." Journal of Operations Research Society, Vol. 6, February, 1958, pp. 35-55.

51. Karush, W., and A. Vazsonui. "Mathematical Programming and Service Scheduling," Journal of the Institute of Management Sciences, Vol. 3, January, 1957.

52. Kuratani, Yoshiro, and Nelson. "A Pre-computational Report on Job-Shop Simulation Research," Management Sciences Research Project, University of California, Los Angeles, October, 1959.

53. Kuratani and McKenny. "A Preliminary Report on Job Shop Simulations Research," Management Sciences Research Project, University of California, Los Angeles, March, 1958.

54. Little, J. D. C., D. W. Sweeney, C. Karel. "An Algorithm for the Traveling Salesman Problem," Journal of Operations Research Society, Vol. 11, 1963, pp. 972-989.

55. Lomnioki, Z. A. "A Branch-and-Bound Algorithm for the Exact Solution of the Three-Machine Scheduling Problem," Operations Research Quarterly, Vol. 16, March, 1965, pp. 89-100.

56. Manne, A. S. "On the Job-Shop Scheduling Problem," Journal of Operations Research Society, Vol. 8, March-April, 1960, pp. 219-223, and also in Chapter 12 in reference (61).

57. McNaughton, R. "Scheduling with Deadlines and Loss Functions," Journal of the Institute of Management Sciences, Vol. 5, January, 1959, pp. 1-22.

58. Mitten, L. G. "Sequencing n Jobs on Two Machines with Arbitrary Time Lags," Journal of the Institute of Management Sciences, Vol. 5, April, 1959, pp. 293-298.

59. Mitten, L. G. "A Scheduling Problem," Journal of Industrial Engineering, Vol. 10, March, 1959, pp. 131-135.

60. Muth, J. F. "The Effect of Uncertainty in Job Times on Optimal Schedules," Chapter 18 in reference (61).

61. Muth, J. F., and G. L. Thompson. Industrial Scheduling. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1963, Chapter 12.

62. Nelson, R. T. "Priority Function Methods for Job-Lot Scheduling," Management Sciences Research Project, University of California, Los Angeles, Discussion Paper No. 51, February 24, 1955.

63. Page, E. S. "An Approach to Scheduling Jobs on Machines," Journal of Royal Statistics Society B, Vol. 23, 1961, pp. 484-492.

64. Palmer, D. S. "Sequencing Jobs Through a Multi-Stage Process in the Minimum Total Time--A Quick Method of Obtaining a Near Optimum," Operations Research Quarterly, Vol. 16, March, 1965, pp. 101-107.

65. Reinitz, R. C. "An Integrated Job Shop Scheduling Problem," Ph.D. Thesis, Case Institute of Technology, Cleveland, Ohio, 1961.

66. Reinitz, R. C. "On the Job-Shop Scheduling Problem," Chapter 5 in reference (61).

67. Roy, B. "Cheminement et Connexite dans les graphes - Applications aux problemes d'ordonnancement," METRA, Serie Speciale No. 1, 1962, Societe d'economie et de mathematiques appliquees, Paris.

68. Rowe, A. J. "Toward a Theory of Scheduling," Journal of Industrial Engineering, Vol. 11, March, 1960, pp. 125-136.

69. Rowe, A. J. "Sequential Decision Rules in Production Scheduling," General Electric Company, October, 1958.

70. Rowe, A. J., and J. R. Jackson. "Research Problems in Production Routing and Scheduling," Research Report No. 46, University of California, Los Angeles, October, 1956.

71. Salveson, M. E. "A Problem in Optimal Machine Loading," Journal of the Institute of Management Sciences, Vol. 2, April, 1956, pp. 232-260.

72. Salveson, M. E. "A Computational Technique for the Scheduling Problem," Journal of Industrial Management, Vol. 13, January, 1962, pp. 30-41.

73. Sasieni, Yaspan, and Friedman. Operations Research: Methods and Problems. New York: John Wiley and Sons, Inc., 1959.

74. Sherman, G. R. "The Use of a Computer for Scheduling Students," Speech before annual meeting of Operations Research Society of America, Washington, D. C., May 14-15, 1959.

75. Sisson, R. L. "Machine Shop Simulation Using SWAC: Part II of a Proposal," Management Sciences Research Project, Research Paper No. 58, May, 1956.

76. Sisson, R. L. "Sequencing Theory," Chapter 7 in reference (1).

77. Sisson, R. L. "Method of Sequencing in Job-Shop--A Review," Journal of Operations Research Society, Vol. 7, January, 1959, pp. 10-29.

78. Smith, W. E. "Various Optimizers for Single-stage Production," Naval Research Logistics Quarterly, Vol. 3, March and June, 1956, pp. 59-66.

79. Smith, W. E. "Application of a Posteriori Probability," Management Sciences Research Project, Research Report No. 56, University of California, Los Angeles, September, 1958.

80. Story, A. E., and H. M. Wagner. "Computational Experience with Integer Programming for Job-Shop Scheduling," Chapter 14 in reference (61).

81. Szwarc, W. "Solution of the Akers-Friedman Scheduling Problem," Journal of Operations Research Society, Vol. 8, November, 1960, pp. 782-788.

82. Thompson, G. D. "Recent Developments in the Job Shop Scheduling Problem," Naval Research Logistics Quarterly, Vol. 7, December, 1960, pp. 585-589.

83. Wagner, H. M. "An Integer Linear Programming Model for Machine Scheduling," Naval Research Logistics Quarterly, Vol. 6, June, 1959.

84. White, C. R. "An Algorithm for Finding Optimal or Near Optimal Solutions to the Production Scheduling Problem," Ph.D. Thesis, Purdue University, Lafayette, Indiana, 1963.

JOB-SHOP SCHEDULING

by

BALRAJ SINGH SONDHU

B. Sc. (Mechanical Engineering),
Panjab University, 1964

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Industrial Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1967

This report considers the problem of scheduling n jobs, not all identical, on m machines. The sequence of jobs through each machine and the processing times for all the jobs are known. The historical method of Gantt charts is discussed and algorithms are given which amplify the use of Gantt charts such that the scheduling becomes dynamic in nature and some objective function is optimized. Four different approaches, i.e., graphical, combinatorial, integer linear programming, and schedule algebras are presented. Accurate solutions are available for problems not involving more than two or three machines. Of course, there is no limit to the number of jobs. The general problem has been treated under severely simplifying assumptions and the solutions are only approximate. Moreover, the formulation of the problem is incomplete. Different functions of time have been optimized but an objective function involving cost or profit should be optimized.

Even under the simplifying assumptions the amount of computation required for solving a problem of reasonable size is large. With the new high-speed computing equipment, it is possible to solve real life problems of scheduling. So far the most fruitful approach to the complex scheduling problems which occur in reality seems to be that of simulation technique. The fruitfulness of this approach also is enhanced by the availability of high-speed computing equipment.